Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Sverre Graffer, Pål K. Fosmo, Sacit Ali Senkaya

Observability and optimizing sensor placement in a maglev system

Masteroppgave i Industriell kybernetikk
Veileder: Damiano Varagnolo
Medveileder: Hans A. Engmark
Juni 2024

NTNU
Kunnskap for en bedre verden

Sverre Graffer, Pål K. Fosmo, Sacit Ali Senkaya

Observability and optimizing sensor placement in a maglev system

Masteroppgave i Industriell kybernetikk
Veileder: Damiano Varagnolo
Medveileder: Hans A. Engmark
Juni 2024

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU
Kunnskap for en bedre verden

P. I. D. K. FOSMO
S. A. SENKAYA
S. GRAFFER
H. A. ENGMARK
D. VARAGNOLO

MAGGY

HANDS-ON CONTROL LEARNING WITH A MAGLEV SYSTEM

DEPARTMENT OF ENGINEERING CYBERNETICS,
NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Contents

Introduction 11

PART I Theoretical Prerequisites 17

1 Mathematical prerequisites 21
2 Physics prerequisites 23
3 Control prerequisites 25

PART II Hardware 31

4 A brief history on Maglev 35
5 Introducing Maggy 39
6 Making your own Maggy 41

PART III Software 45

7 Getting ready for programming with Arduino 49
8 System setup & testing 51
9 Serial communication for off-chip computations 55

PART IV Control 59

10 Physics-driven modeling 63
11 Data-driven modeling 67
12 State observers 83
13 Stabilizing the system 101
14 Controller implementation 129
15 Concluding remarks: Stability and controller implementation 147

PART V Magrobatics & Beyond 149

16 The future of Maggy 153

Bibliography 155

List of Figures

1 An illustration of V2.6 of Maggy. 11
2 An illustration of the Hoshonchu airbonsai garden [Hoshinchu, 2024]. 12

1.1 Example of minor matrix part of A 22

3.1 Illustration of three different systems 26
3.2 Lyapunov function energy cone visualization 27

5.1 An illustration of Maggy. 39
5.2 Image of Levitation of Maggy V2.0. 39

7.1 Arduino preferences window 50
7.2 Arduino library manager 50

8.1 The "Port"-menu with Teensy selected 52
8.2 The Arduino serial plotter 53

9.1 UART packet bit allocation 56

11.1 Comparison of simulated states against their reference values 77
11.2 Closed-loop response obtained with the real system 78
11.3 Closed-loop response obtained with the DSR method 79
11.4 Closed-loop response obtained with the *N4SID* method 79
11.5 Closed-loop response obtained with the *MOESP* method 80

12.1 System configuration 90
12.2 x, y (states) displayed as x, y (plot), z = zEq - 0.002 m 90
12.3 x, y (states) displayed as x, y (plot), z = zEq 90
12.4 x, y (states) displayed as x, y (plot), z = zEq + 0.002 m 91
12.5 α, β (states) displayed as x, y (plot), z = zEq 92
12.6 The states of interest used for calculating an average. z = zEq. All the other state variables are zero. This is similar to the heat map 12.3. 93
12.7 Shows observability vs the distance from the two off-center sensors to the z-axis. 94
12.8 Shows observability vs the distance from the two off-center sensors to the z-axis. One curve for every relative angle. 95

12.9 Shows observability vs the distance from sensors z coordinates. 95
12.10 Shows observability vs one off center sensor rotated by this angle around the z-axis compared to its normal position. 96

13.1 Singular values of controllability matrix 103
13.2 An illustration of the system with axes. 103
13.3 Illustration of levitating magnet 104
13.4 PID simulation with no radial deviation 106
13.5 PID simulation with radial deviation 106
13.6 Plot for equilibrium deduction 108
13.7 LQR simulation with no radial deviation 108
13.8 LQR simulation with radial deviation 109
13.9 Simulink setup 119
13.10 Stability test illustration 120
13.11 Magnetic field heatmap w/o input 121
13.12 Magnetic field heatmap w/ input 121
13.13 Circumference test points illustration 123
13.14 ROC comparison: LQR and FL 124
13.15 ROC comparison: FL with constrained input 125
13.16 ROC comparison: FL with time delay 126
13.17 Illustration of the earlier configuration 127
13.18 Illustration of the magnet configuration of Maggy V2.6 128

14.1 MATLAB implementation structure 131
14.2 Matlab implementation timed portions 135
14.3 Matlab implementation running time 136
14.4 Input signal u calculated by MATLAB 137
14.5 Input signal u as received by the Teensy 138
14.6 Arduino implementation structure 139
14.7 MATLAB mldivide decision tree 141
14.8 Arduino implementation timed portions 143
14.9 Arduino implementation running time 143
14.10 Simulink diagram with proposed changes 146

List of Tables

11.1 Variance accounted for between the real output and closed-Loop responses from different estimation methods, given as percentage 80

12.1 Method parameters 89
12.2 Average of region in selected heat maps 91
12.3 Testing different combinations of different states to include. Measurements from all three sensors are included in this test. 98
12.4 Testing different combinations of different states to include. 98

13.1 Different pole configurations for different input constraints. 118

14.1 Memory size for some useful data types 134
14.2 Table detailing the boundaries set for the random generation of state estimations 135

Dedicated to all the co-workers, students, friends, and family who have brought Maggy to life.

Introduction

Why this book?

This book represents our first edition of a complete description of *Maggy*, a *take-home lab* developed at the Department of Engineering Cybernetics (ITK) at the Norwegian University of Science and Technology (NTNU).

We believe that practical lab exercises are an essential part of the learning journey of any aspiring control engineer, as they play a key role in self-efficacy when transitioning from a university student to a working engineer. However, not every student has access to sufficient lab facilities at their home universities. A *take-home lab* is a concept that we think can help students in these kinds of situations. The namesake of this book, *Maggy*, represents our first stab at the take-home lab concept.

Take-home lab: teaching material that is developed with the intention for students to work on it from anywhere, providing the same learning benefits as a normal on-site lab.

What is Maggy?

Maggy is a small and portable magnetic levitation system, designed and assembled from the ground up by students at NTNU. Figure 1 shows version 2.6 of Maggy; one of several iterations in the development, and as of the writing of this book, the latest viable system.

All iterations of Maggy consist of a base (the green PCB), to which a microcontroller, sensors, some electronics, permanent magnets (stacked gray magnets), and electromagnets (copper coils) are mounted. The goal of Maggy is to actively adjust the current in the electromagnets to enable the large permanent magnet seen in the figure to hover above the platform.

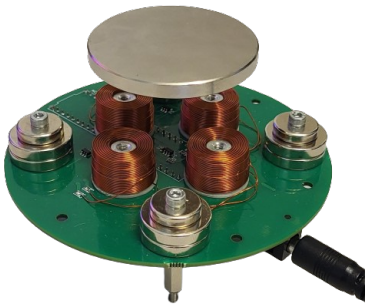


Figure 1: An illustration of V2.6 of Maggy.

The concept of Maggy is not new; there already exist many commercial products that use the same kind of system configuration to make things hover (see for instance Figure 2). What is new about

Maggy is how it is specifically tailored towards teaching, with the system description of Maggy being fully available to students.

Figure 2: An illustration of the Hoshonchu airbonsai garden [Hoshinchu, 2024].

[1] https://github.com/Hansolini/Take-home-Maglev-lab

How to read this document

This book contains a complete description of *Maggy*, all the way from how to build and assemble the hardware required, to the implementation of advanced control methods. It is intended to be read by students and teachers who would like to learn more about the system; be it understanding how it works, building one themselves, or tailoring lab exercises to fit into their curriculum.

The text content of this book is only half the story; our GitHub repository[1] is the other half, and should be considered as much part of the "book" as the text itself. The GitHub repository also contains the most up-to-date information on the development of the system.

The book is divided into four parts, each providing context for the different aspects of the Maggy lab. The sections can be read all together or independently, and are divided as follows:

Part I covers the essential prerequisites in mathematics, physics, electronics, programming, and control theory.

Part II delves into the hardware aspects of Maggy, including its history and assembly instructions.

Part III focuses on the software setup, programming, and system testing.

Part IV introduces control methods, modeling, and implementation strategies for Maggy.

Part V is a discussion on the future of the Maggy, its construction, application, and control capabilities.

Contributions of the authors

The first three authors of this book have written it in fulfillment of their Master's thesis in Cybernetics and Robotics at ITK, NTNU, under the supervision of the last two authors. Instead of following the traditional approach of writing three separate theses, we have collaborated on this book from the very beginning of the project work. In mind we kept an ultimate goal of facilitating the uptake of both practical and theoretical knowledge future students learning control. In these months we have been then striving for, using a sentence we recorded during our first meeting,

developing a new take-home lab that may enable students in the field of electrical, mechanical, and control engineering learn and become proficient professionals with hands-on competences.

For evaluation purposes, we clarify the contributions of each author as follows:

- **Pål Ivar Delphin Kværnø Fosmo**

  1 Linear algebra
  2 Physics prerequisites
  3 Control prerequisites
  III Software
  13 Stabilizing the system
  14 Controller implementation
  15 Concluding remarks: Stability and controller implementation

- **Sverre Graffer**

  12 Observability

- **Sacit Ali Senkaya**

  10 Physics-driven modeling
  11 Data-driven modeling

A large part of their contributions is the development of relevant code for the project. These contributions are available through the GitHub repository.

List of abstracts

For the sake of completeness and compliance with the requirements of what is expected to be delivered as a thesis at NTNU, the following section contains three abstracts in accordance with the contributions outlined above.

Pål Ivar Delphin Kværnø Fosmo

This thesis presents the development of an advanced controller for a magnetic levitation (maglev) system that is called Maggy and is intended for educational purposes. The primary objective of Maggy is to enable students to learn, through facing the problem of stabilizing a levitating magnet, different control strategies in a hands-on approach to learn control theory. Maggy has multiple properties that make controlling it a unique challenge while also being suitable for exploring different modes of control, both traditional and more advanced ones.

The thesis introduces central concepts related to magnetic levitation and control, as well as brief introductions to programming microcontrollers and serial communication theory in Parts I and III. It is divided into two main parts, one detailing the development of a controller and analysis of system stability. The other part is solely focused on exploring different modes of implementation for the controller and choosing a suitable approach to provide the best basis for stability for the controller. More specifically, it details analyses of different aspects, such as stability and implementation methods of a feedback linearizing controller, as well as brief introductions to more conventional control methods such as PID and LQR. Through experimentation, their performances are measured relative to each other and used as a reference for assessing the performance of the feedback linearizing controller.

Development and experiments are mainly done through simulation in Simulink using a pre-developed simulator in MATLAB, but part of the implementation analysis includes microcontroller programming. The findings highlight the system's controllability and stability and the potential for interactive and engaging student assignments in control engineering education.

Sverre Graffer

This thesis is dedicated to the generic problem of inferring the state of a system starting from noisy information from sensors and a intrinsically imperfect knowledge of the equations governing the system – i.e., a model of it.

Many types of controllers require that estimates of the internal states of the system are available. This requires thus implementing an observer, and for an observer to work, the system must be observable.

A first step towards implementing working observers is thus characterizing such observability properties - something that may depend on the physical properties of the system under consideration.

This thesis is then dedicated to the process of characterizing and testing such observability properties, and offers the reader the occasion of learning how to do it on the practical case of a maglev system. The text accompanies thus the learner in understanding if using an observer is feasible for further control purposes. This work focuses especially on a specific method from the literature to test the observability of the maglev system. The text takes then to practical results indicating that the system is probably observable enough, in the sense that observing the internal states of the system using an

observer might be of practical use for state-feedback based control techniques.

The last part of the thesis discusses also how analysing the observability properties of a system is actually a process that may help redesigning the system itself, something that is here exemplified through the process of optimizing the placement of the sensors of Maggy.

Sacit Ali Senkaya

This thesis explores how system identification for closed loop systems, that crucial methodology in automatic control that enables modeling dynamic systems based on observed input-output data, may be learned in a hands-on fashion by means of Maggy. The thesis begins with a foundational explanation of parametric and nonparametric modeling approaches, highlighting their advantages and disadvantages, and the computational requirements that come together with the associated algorithms. The core of the text discusses then various identification methods, including the deterministic-stochastic realization (DSR), numerical algorithms for subspace state space system identification ($N4SID$), and multivariable output-error state space (MOESP) methods. The thesis dedicates then some space in highlighting their applications in both open-loop and closed-loop systems.

A significant focus is also placed on the practical implementation of these methods to estimate the state-space models. In doing so this thesis also introduces the concept of the Prediction Error Method (PEM) for refining system models, and discusses the use of subspace identification techniques to enhance model accuracy without complex optimization processes.

Ultimately, this thesis serves as a comprehensive guide that not only discusses the theoretical aspects of system identification but also provides a detailed guideline for its practical application, making it a valuable resource for learners that aim at becoming proficient engineers and researchers in the field of control systems.

Part I

Theoretical Prerequisites

Introduction

In this part of the book, we will cover the fundamental theoretical prerequisites necessary to gain a complete understanding of Maggy, the maglev system. This is particularly important for the understanding of the first-principles model of Maggy, presented in Part IV.

This part is divided into three chapters, organized as follows:

Chapter 1 provides an overview of the most important mathematical topics encountered when working with Maggy, such as fundamental calculus and linear algebra.

Chapter 2 is an introduction to the major physical phenomenon present in Maggy, mainly electromagnetism and classical mechanics.

Chapter 3 is an overview of control-theoretic concepts relevant for the stability and operation of Maggy.

All chapters assume a basic understanding of the topics at an undergraduate engineering level. We also remark again that this book is under continuous development, hence the topics covered here are only what is relevant for the understanding of the theses of the three main authors.

Chapter 1

Mathematical prerequisites

Calculus

Linear algebra

Some basic understanding of matrix decomposition is needed for the controller development. The first one is called *singular value decomposition* and is briefly used in the controllability analysis. Singular value decomposition is a method of factorizing a matrix A into one diagonal and two orthogonal unitary matrices, given that A is a real matrix:

$$A = U\Sigma V^T. \tag{1.1}$$

Σ contains the *singular values* of matrix A and can be likened to a scaling factor for its associated unitary vectors in the transformed coordinate system. A vector S denotes a gathering of the values along the diagonal of Σ and can be used as a measure for the rank of a matrix.[1]

[1] Kenneth Lange. Singular value decomposition. *Numerical analysis for statisticians*, pages 129–142, 2010

QR decomposition is a method that can be used to solve linear systems on the form $Ax = b$ by decomposing the A matrix like

$$A = QR,$$

where Q is an orthonormal matrix and R is an upper rectangular matrix. This transforms the system to

$$QRx = b, \tag{1.2}$$

using the fact that $Q^T = Q^{-1}$ gives

$$Rx = Q^T b. \tag{1.3}$$

The upper rectangular structure of R makes it easy to solve this system for b recursively.

The main issue is how we implement this method. In our mission to develop a QR decomposition to employ on the Teensy, two approaches were investigated, namely *Gram-Schmidt* and *Householder transformations*. Gram-Schmidt relies on using projections of combinations of the column vectors to form normalized and orthogonal vectors that can be combined to find a Q matrix. R is then found by

$$R^T Q^T = A^T. \tag{1.4}$$

The classical Gram-Schmidt method is numerically unstable, so special considerations must be made to ensure that Q is actually orthogonal. Therefore, another method is also examined in relation to the implementation.

The Householder transformation uses Householder reflections, which can reflect a vector x about a hyperplane orthogonal to a unit vector v by multiplying it with a *Householder matrix* P. The Householder matrix P is symmetric and orthogonal and given by

$$P = I - \frac{2vv^T}{v^T v}. \tag{1.5}$$

The reflection can be summarized as wanting

$$Px == x - \frac{2vv^T}{v^T v}$$

to to be a multiple of $e_1 = I_m(:, 1)$. Defining $v = x + \alpha e_1$ allows for the reflection to become a multiple of e_1 by setting $\alpha = ||x||_2$, yielding

$$Px = ||x||_2 e_1. \tag{1.6}$$

By recursively solving (1.5) for P, where v is found by defining x as the current column vector from A, and $\alpha = ||x||_2$. For the sake of the QR decomposition, this P becomes Q_1, and by multiplying it with the original A, the first column vector is transformed to $||x||_2 e_1$. This procedure is repeated on all the minor matrices not yet solved in A, as exemplified in Figure 1.1. Note that H_i in this figure is synonymous to our Q_i.

$$H_2 H_1 A = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \boxed{\mathbf{\times}} & \boxed{\times} & \boxed{\times} \\ 0 & 0 & \boxed{\mathbf{\times}} & \boxed{\times} & \boxed{\times} \\ 0 & 0 & \boxed{\mathbf{\times}} & \boxed{\times} & \boxed{\times} \\ 0 & 0 & \boxed{\mathbf{\times}} & \boxed{\times} & \boxed{\times} \end{bmatrix}$$

Figure 1.1: Example of a *minor* of the A matrix not yet solved after two iterations. The new vector to be solved for is marked in bold, Golub and Van Loan, 2013.

The methods and theory outlined about QR decompositions are further detailed in Golub and Van Loan, 2013, and to some extent Gander, 1980.[2,3]

[2] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. JHU press, 2013

[3] Walter Gander. Algorithms for the qr decomposition. *Res. Rep*, 80(02): 1251–1268, 1980

Chapter 2

Physics prerequisites

The following section on electromagnetism is taken from the specialization project report by Pål Ivar Delphin Kværnø Fosmo.[1]

[1] Pål Ivar Delphin Kværnø Fosmo. Control of a magnetic levitation system using feedback linearization. 2023

Electromagnetism

One of the main aspects of the controller design is working with magnetic fields and altering them to allow for the levitation of a free magnet. Some general knowledge about these fields is therefore needed to understand the system's dynamics.

Biot-Savart's law

Biot-Savart's law is central for calculating and describing the magnetic field generated by the solenoids. The law provides an equation relating the generated magnetic field to a constant electric current. This relationship is given by

$$B(\boldsymbol{r}) = \frac{\mu_0}{4\pi} \int_C \frac{I d\ell \times r'}{|r'|^3}, \tag{2.1}$$

where r denotes the position in 3D space and I the current in the wire. μ_0 is the permeability of air, $d\ell$ is a vector along the the wire C while r' is the displacement vector from the wire element to the point in which the field is calculated.[2]

[2] I. S. Grant. *Electromagnetism*. Wiley, 1990. ISBN 978-0-471-92712-9

In this system the solenoids are approximated as single thin wire loops to reduce computational times. The law's application in this system is therefore akin to the one described in Cárdenas and González, 2021. The simplification reads:[3]

[3] Migdonio A. González and Dorindo E. Cárdenas. Analytical expressions for the magnetic field generated by a circular arc filament carrying a direct current. *IEEE Access*, 9:7483–7495, 2021. DOI: 10.1109/ACCESS.2020.3044871

$$\boldsymbol{B}(\phi, \rho, z) = \begin{bmatrix} \phi \\ -\frac{z}{\rho} ck \left[\mathrm{K}\left(k^2\right) - \frac{\rho^2+R^2+z^2}{(\rho-R)^2+z^2} \mathrm{E}\left(k^2\right) \right] \\ ck \left[\mathrm{K}\left(k^2\right) - \frac{\rho^2-R^2+z^2}{(\rho-R)^2+z^2} \mathrm{E}\left(k^2\right) \right] \end{bmatrix} \tag{2.2}$$

$$k^2 = \frac{4R\rho}{(R+\rho)^2 + z^2}$$
$$c = \frac{\mu_0}{4\pi\sqrt{R\rho}} NI.$$

Where $K(k^2)$ and $E(k^2)$ denote the complete elliptical integrals of the first and second kind. The polar coordinates (ϕ, ρ, z) represent the distance vector from the point in space to the center point of the solenoid.

Newton's laws of motion

The following section on forces and torque is taken from the specialization project report by Pål Ivar Delphin Kværnø Fosmo.[4]

[4] Pål Ivar Delphin Kværnø Fosmo. Control of a magnetic levitation system using feedback linearization. 2023

Forces and torque

The system's translation and rotation are modeled using Newton-Euler Equations of Motion. This means that the calculations of the magnetic fields must be translated into forces and torques exerted on the levitating magnet. Expressions for the force and torque acting on a permanent magnet of zero size in a uniform magnetic field can be written as:

$$\boldsymbol{F} = \nabla \left(\frac{\boldsymbol{J}}{\mu_0} \cdot \boldsymbol{B} \right) \tag{2.3}$$
$$\boldsymbol{\tau} = \frac{\boldsymbol{J}}{\mu_0} \times \boldsymbol{B} \tag{2.4}$$

where $\boldsymbol{J}$ denotes the magnetic polarization of the magnet.[5]

[5] Matthew N. O. Sadiku. *Elements of Electromagnetics*. Oxford University Press, USA, 2001. ISBN 019513477X

By integrating this formula over a volume, the force and torque on an arbitrary permanent magnet can be calculated. Ravaud et al., 2010, proposes an *equivalent magnetization approach* for the models of the permanent magnets, where the magnetization surface current **K** can be expressed as[6]

[6] R. Ravaud, G. Lemarquand, S. Babic, V. Lemarquand, and C. Akyel. Cylindrical magnets and coils: Fields, forces, and inductances. *IEEE Transactions on Magnetics*, 46(9):3585–3590, 2010. DOI: 10.1109/TMAG.2010.2049026

$$\boldsymbol{K} = \frac{\boldsymbol{J} \times \boldsymbol{n}}{\mu_0} \tag{2.5}$$

which allows for the forces and torques to be accurately calculated by the surface integrals over the surface **S** of the magnets

$$\boldsymbol{F} = \int_S \boldsymbol{K}(\alpha, \beta, \gamma) \times \boldsymbol{B}_{Base}(x, y, z) dS, \tag{2.6}$$
$$\boldsymbol{\tau} = \int_S \frac{1}{\mu_0} \boldsymbol{J}(\alpha, \beta, \gamma) \times \boldsymbol{B}_{Base}(x, y, z) dS. \tag{2.7}$$

Chapter 3

Control prerequisites

This chapter briefly introduces central concepts related to stability and system control.

Stability

Stability is a fundamental property of dynamical systems, which tells us how a system can be expected to behave under certain circumstances. Conceptually, it only means that the states inherent in the system do not diverge towards infinity but are confined to within a smaller portion of the state space. For the linear case, stability can be divided into *input-output* stability and *state space* stability. *Input-output* stability says something about how the output reacts to an input. Being input-output stable means that the system responds with a bounded output when excited by a bounded input.[1]

[1] Trond Andresen, Jens G. Balchen, and Bjarne A. Foss. *Reguleringsteknikk*. Institutt for teknisk kybernetikk, NTNU, 2016. ISBN 9788278422021

State space stability is related to the behavior of the states within the state space themselves. A system is said to be state space stable if the states converge after being excited by a bounded input signal. This is the type of stability most relevant to Maggy. Stability is often examined by looking at the poles and eigenvalues of a linear system. Maggy is a nonlinear system, which means that these model parameters are not readily apparent in the same way as for a linear system. Therefore, alternative methods must be employed to assess the stability of this system. Figure 3.1 illustrates the concept of stability through three different systems, all with different stability properties, and their behavior within the state space z_1, z_2. System (a) is stable since it converges towards a point in the state space, while (b) is unstable because it diverges from the point. System (c) is *marginally stable* since it does not converge, but it does not diverge either.

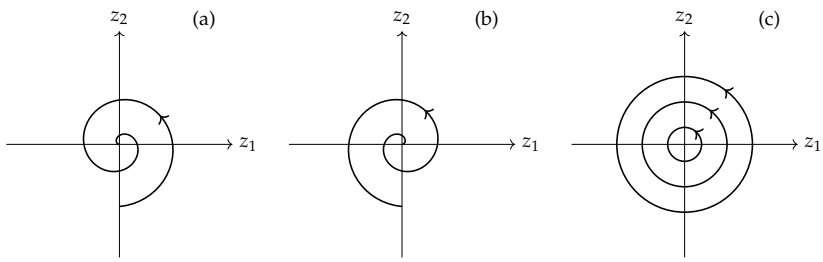


Figure 3.1: Illustration of three different systems, Khalil, 2002.

Stability is rarely inherent in real systems. Therefore, stability must be enforced on the system through the use of controllers that can manipulate the input signal to "force" the system towards a specific target and keep it there stably.

Lyapunov stability theory

As covered earlier, linear and nonlinear systems differ quite substantially when it comes to stability analysis. A central concept in the field of nonlinear system analysis is *Lyapunov stability*. In his 1892 paper *The General Problem of the Stability of Motion*, Aleksandr Lyapunov outlined a method for analysis of a system's stability through the usage of a so-called *Lyapunov function*.[2]

[2] Shankar Sastry. *Nonlinear Systems: Analysis, Stability, and Control*, chapter 5. Springer New York, 1999. ISBN 978-1-4757-3108-8

This method revolves around determining a system's energy levels, with the Lyapunov function being analogous to the potential function of classical dynamics. The method aims to analyze the stability of equilibrium points, first of all locally but potentially also globally. [3,4] It is helpful to briefly introduce some central definitions and notions as featured in Khalil, 2002 and Sastry, 1999 that are relevant to Maggy.

[3] Shankar Sastry. *Nonlinear Systems: Analysis, Stability, and Control*, chapter 5. Springer New York, 1999. ISBN 978-1-4757-3108-8

[4] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002. ISBN 0130673897

Equilibrium point: A point in the state space that the system seeks to converge to, if the system starts in this point, it will stay there indefinitely. In Lyapunov terms, an "energy minimum". A system can have no such points, one or multiple.

Lipschitz continuity: A property of a system wherein its derivative is bounded. This means that it at no point has an infinite rate of growth. It can be either local or global, depending on where the Lipschitz condition holds.

Asymptotic stability: Stability itself only implies that the system does not diverge. Asymptotic stability implies that the system *will* return to an equilibrium point as $t \rightarrow \infty$, but does not give a convergence rate. This can be expanded to the local and global scope, depending on the number of equilibrium points and their placements.

Radial unboundedness: A function $f(x)$ is radially unbounded if and only if $||x|| \rightarrow \infty$ then $f(x) \rightarrow \infty$.

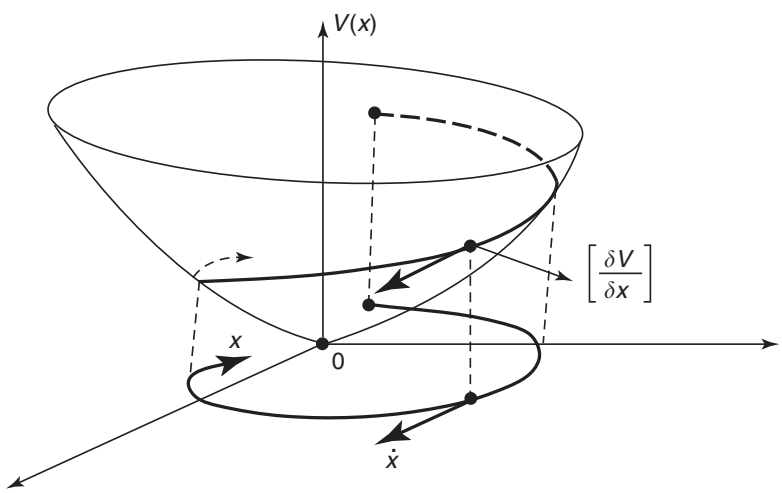


Figure 3.2: Illustration of a Lyapunov function as an energy cone, with a stable state moving towards an equilibrium [Chen, 2004]

The main idea behind a Lyapunov analysis is choosing an energy analogous function, called a candidate function, on which the analysis will be carried out. The primary goal is to establish its validity as a full *Lyapunov function*, whose properties will reflect the system's stability. For this candidate function to become a full Lyapunov function, it must fulfill the following criteria:

1. $V(0) = 0$
   $V(x) > 0$ for all $x \in \mathbb{D} \setminus \{0\}$
2. $\dot{V}(0) = 0$
   $\dot{V}(x) \leq 0$ for all $x \in \mathbb{D} \setminus \{0\}$
3. A Lyapunov function becomes a *strict* Lyapunov function if
   $\dot{V}(0) = 0$

$\dot{V}(x) < 0$ for all $x \in \mathbb{D} \setminus \{0\}$

within some domain $\mathbb{D}$.[5]

[5] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002. ISBN 0130673897

This paves the way for the definitions of *Lyapunov's direct method*

- If $\exists$ a Lyapunov function for $x = 0$, then $x = 0$ is stable
- If $\exists$ a strict Lyapunov function for $x = 0$, then $x = 0$ is asymptotically stable.

This proves local stability. In order to prove asymptotic stability for the entire domain, another criterion must be fulfilled

- If $\exists$ a strict Lyapunov function for $x = 0$, and V is radially unbounded, then $x = 0$ is globally asymptotically stable.

PID

As far as automatic control of dynamic systems goes, the *PID* controller is by far the most widely used feedback controller, used in 90% of industrial control applications.[6]

Transient errors: errors related to oscillations in a system when controlled to a set-point.

[6] Ramon Vilanova and Antonio Visioli. *PID control in the Third Millennium : lessons learned and new approaches*. 2012

Carl Knospe, 2006 defines a PID controller as a controller that fulfills three common requirements for general control problems:

1. A proportional part (P) that can react immediately to any errors,
2. an integral part (I) that helps bring the steady-state error to zero over time, and
3. a derivative part (D) that helps combat transient errors.

A PID controller can be implemented in two configurations, *parallel* and *series* type. The difference between the two lies in how the tuning parameters impact each portion of the controller.

$$u(t) = K_c \left(e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{de(t)}{dt} \right) \tag{3.1}$$

The format of a series-type PID controller is shown in (3.1), where $e(t)$ denotes the error between the reference signal and the actual state of the system. The series type is known as an *ideal representation* since all portions of the controller are separated and only impacted by the gains and time constants associated with themselves. The controller is tuned by changing the gain and time constants K_p, T_i, and T_d [7].

[7] Rakesh P. Borase, D. K. Maghade, S. Y. Sondkar, and S.N. Pawar. A review of pid control, tuning methods and applications. *International Journal of Dynamics and Control*, 9:818–827, 2020. DOI: https://doi.org/10.1007/s40435-020-00665-4

The main advantage of employing a PID controller in a system is its flexibility. Contrary to model-based controllers, a PID controller does not rely on knowledge of the system it controls, which makes it very applicable to various real-world systems. On the other hand, this makes tuning the controller challenging, given that the optimal parameters often depend on the nature of the system. Therefore, tuning is often done through empirical methods such as *Ziegler-Nichols* methods and *SIMC*.[8] In the nonlinear case, these methods become hard to apply and while some methods can be used for nonlinear systems, the PD controller presented in Part IV was tuned through trial and error.[9]

[8] Trond Andresen, Jens G. Balchen, and Bjarne A. Foss. *Reguleringsteknikk*. Institutt for teknisk kybernetikk, NTNU, 2016. ISBN 9788278422021

[9] Rakesh P. Borase, D. K. Maghade, S. Y. Sondkar, and S.N. Pawar. A review of pid control, tuning methods and applications. *International Journal of Dynamics and Control*, 9:818–827, 2020. DOI: https://doi.org/10.1007/s40435-020-00665-4

LQR

Another important class of controllers is the model-based controllers, which can offer a more robust control scheme, given that they are directly deduced using the model parameters. The LQR controller is a model-based controller closely associated with optimal control. The LQR controller is a state feedback controller, and like the PID, it relies on a feedback loop to receive real-time updates about the system. Through a control law $u = -Kx$, the gain matrix K generates an appropriate input u to stabilize the system.[10] It bases itself on a simple principle: to minimize the *control signal energy* through weight matrices. In essence, this can be summed up by the equation

[10] Joao P. Hespanha. *Linear systems theory*. Princeton university press, 2018

$$\int_0^\infty y(t)^T Q y(t) + u(t)^T R u(t) dt \tag{3.2}$$

The goal is to minimize this integral by tuning the positive definite weight matrices Q and R. The relation between the two is the main contributing factor to the system's behavior. With $Q >> R$, the most efficient way to achieve this is by employing a significant input to minimize the output, and vice versa when $R >> Q$.[11]

[11] Joao P. Hespanha. *Linear systems theory*. Princeton university press, 2018

This forms the basis for the LQR algorithm, in which the algebraic Riccati equation

$$A^T P + PA + C^T QC - PBR^{-1}B^T P = 0 \tag{3.3}$$

is solved to obtain a matrix P, which can be used to find a gain matrix

$$K = R^{-1}B^T P \tag{3.4}$$

that stabilizes the system while minimizing the LQR criterion outlined in (3.2).[12]

[12] Joao P. Hespanha. *Linear systems theory*. Princeton university press, 2018

Feedback linearization

Feedback linearization is a tool for transforming a nonlinear system into a linear one without approximating the system model. This allows for the use of linear controllers without the problems related to modeling errors that are introduced when using traditional linearization techniques. The main idea behind this approach is to cancel the nonlinearities and impose a linear structure on the system by using the input.

Generally, designing an input u that can directly cancel all nonlinearities in a system might prove difficult in many cases. If a system can be presented in a general semi-linear form as

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\gamma(\boldsymbol{x})(u - \alpha(\boldsymbol{x})). \tag{3.5}$$

The general input then takes the form

$$u = \alpha(\boldsymbol{x}) + \beta(\boldsymbol{x})v \tag{3.6}$$

where $\beta(\boldsymbol{x}) = \gamma^{-1}(\boldsymbol{x})$ and v denotes the desired input or linear control law.[13]

[13] Hassan K. Khalil. *Nonlinear Systems*. Prentice Hall, 2002. ISBN 0130673897

For a system to be *Input-State Linearizable*, there must exist a transformation $\boldsymbol{z} = \boldsymbol{z}(\boldsymbol{x})$ that transforms the system into a form where all nonlinearities can be removed by the input directly.[14] This transformation will leave the system looking like this

[14] J. E. Slotine and W. Li. *Applied Nonlinear Control*. Prentice Hall, Inc, 1991

$$\dot{\boldsymbol{z}} = A'\boldsymbol{z} + B'\boldsymbol{v} \tag{3.7}$$

where

$$A' = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \dots & \dots & 1 & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, B' = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}. \tag{3.8}$$

Conceptually, this transformation renders all states linear except for the last one, which is the only state directly affected by any input. By examining this last state and designing a fitting input u in line with (3.6), the nonlinearities can be canceled, and the originally nonlinear system can be likened to a linear system.

list of things one should know (with links to other material) to grasp the concepts

best would actually be that there is a list of conceptual questions where the student can test herself/himself, and recommendations of what to study in case one is not able to answer that questions

Part II

Hardware

Introduction

In this part, we present the physical structure and components of Maggy, specifically focusing on version 2.6, the most recent viable design. The chapters aim to offer a comprehensive overview, enabling the reader to assemble their own Maggy, or even modify the system to create a customized version. Frequent references to the GitHub repository, where design files and instructions can be found, are included throughout the different chapters.

This part is divided into three chapters, organized as follows:

Chapter 4 provides historical context and descriptions of various types of magnetic levitation systems, placing Maggy within this context.

Chapter 5 details the components and functions of Maggy itself.

Chapter 6 offers a step-by-step guide for assembling an instance of Maggy.

Chapter 4

A brief history on Maglev

Introduction to Maglev Technology

The word "Maglev" and the concept of magnetic levitation were first proposed in the early 20th century to reduce friction and enable high-speed transportation in trains. These trains float above the tracks, eliminating friction and allowing for extremely high speeds.

Maglev Trains

The first commercial Maglev train, the Transrapid, was developed in Germany and began operations in 1984. Japan's L0 Series and China's Shanghai Maglev are other notable examples, reaching speeds above 600 km/h.

Frictionless Bearings

Another application of Maglev technology is in the development of frictionless bearings. These bearings utilize magnetic fields to suspend moving parts, significantly reducing wear and tear and increasing efficiency and longevity. Frictionless bearings are used in various high-precision devices, including medical equipment, industrial machinery, and scientific instruments.

Planar Motors

Planar motors are a sophisticated application of Maglev technology used in precision manufacturing and robotics. These motors operate on a flat surface and can move in multiple directions without physical contact, offering high precision and flexibility. Planar motors are essential in industries that require precise positioning and high-speed motion, such as semiconductor manufacturing and automated

assembly lines.

Consumer Products

Maglev technology has also made its way into consumer products. Flyte, for instance, is a popular product that uses magnetic levitation to float a light bulb in mid-air. This innovative use of Maglev technology not only serves as a functional light source but also as a captivating piece of modern decor. Other consumer products utilizing Maglev include levitating speakers, planters, and even novelty items like floating globes. Aside from Maglev trains, this is perhaps the most well-known use case of magnetic levitation technology.

NASA Moon Levitation

NASA has more recently announced the planned use of Maglev "roads" for the transportation of material when future moon bases are to be constructed. Similar to planar motors, these roads contain a magnetic base that can be placed directly on the lunar surface, where levitating carts are placed on top. Unlike planar motors and most other commercial uses of magnetic levitation technology, this relies on passive levitation through diamagnetism, which allows for stable levitation without the use of stabilizing forces. This is only possible due to the low gravitational pull of the moon.

Maglev in Education: Quanser and Maggy

Quanser, a leader in engineering education solutions, has integrated Maglev technology into their range of products. They offer a Maglev experiment platform that allows students to study the principles of magnetic levitation and control systems in a hands-on environment. This platform is widely used in universities and research institutions for teaching and research purposes, and it relies on the same principles for levitation as Maggy. However, this platform levitates objects by "hanging" them under a magnet, rendering the problem into a 1D system. Moreover, the Quanser solution is proprietary, and each unit can be very costly, which is a severe hindrance for use as a take-home lab for students without access to expensive lab equipment.

Maggy, our take-home lab system, is designed to bring the benefits of Maglev technology into the educational realm. By providing students with a portable and interactive tool, Maggy enables the exploration of magnetic levitation principles and their applications in

a hands-on manner. This system not only enhances the learning experience but also fosters a deeper understanding of this fascinating technology.

Chapter 5

Introducing Maggy

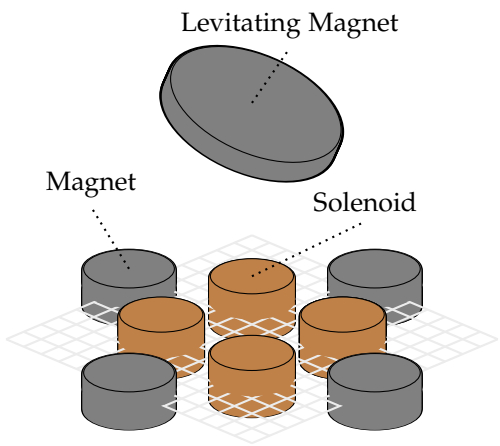


Figure 5.1: An illustration of Maggy.

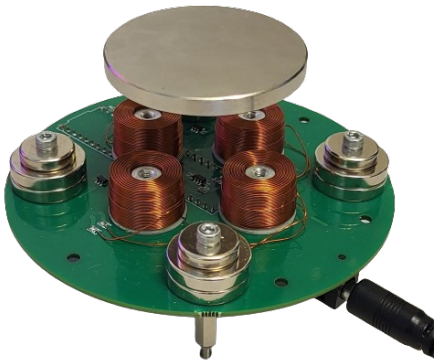


Figure 5.2: Image of Levitation of Maggy V2.0.

Chapter 6

Making your own Maggy

One of the main goals of Maggy is to provide an affordable take-home lab for students without the means or opportunity to learn control concepts in standard university lab sessions. To this end, the following chapter of the book provides a complete physical description of Maggy, with instructions on how to acquire and assemble the system from mostly "cheap" off-the-shelf hardware.

In this chapter, we present a step-by-step guide on how to acquire components and assemble the most recent design of Maggy, version 2.6.

Designing the system

Maggy version 2.6 is designed using KiCad and EasyEDA. The latest blueprints can be found in our GitHub repository under *physical*$_s$*ystem >> hardware >> blueprints*.

System Assembly

Acquire Components

- *Refer to the GitHub repository for the specific components that need to be soldered to the PCB.*
- *Ensure you have the following non-soldered components:*
  - *1 x Teensy 4.1*
  - *4 x solenoids*
  - *8 x permanent magnets (N38 Ø20x4 mm - 3mm hole)*
  - *1 x levitating magnet (N38 Ø50x5 mm)*
  - *8 x 3mm screws and nuts*
  - *(Optional) 4 x 3mm spacers*

Order the PCB

- *Order the PCB from a producer such as JLCPCB, PCBway, or Elprint.*
- *If ordering with assembly, provide the BOM (Bill of Materials) and POS (Pick of Solder) files.*
- *Typically, order with assembly for the bottom side only to reduce costs.*
- *Check the GitHub repository for schematics, Gerber files, and BOM.*

Prepare for Soldering

- *If necessary, receive assistance from the IES Elpro lab for soldering parts that are unavailable in the assembly process.*

Fasten Solenoids

1. *Position the 4 solenoids on the top side of the PCB.*
2. *Secure each solenoid using 3mm screws, inserted from the bottom side of the PCB.*

Solder Solenoid Wires

1. *Identify the wire that exits counter-clockwise from each solenoid.*
2. *Solder this wire to the pad labeled "+" on the PCB.*
3. *Ensure all solenoids are soldered in the same orientation for consistency.*

Prepare Teensy 4.1

1. *Scrape the USB voltage connection on the Teensy 4.1 to disable USB power. This is necessary because the Teensy will be powered externally via a barrel jack plug.*

Fasten Permanent Magnets

1. *Position the 8 permanent magnets as shown in the reference pictures.*
2. *Secure each magnet using 3mm screws inserted from the top, with nuts fastened on the bottom side.*

Connect Teensy 4.1

1. *Align the Teensy 4.1 with the two headers on the back of the PCB.*
2. *Ensure the pins are correctly aligned and securely connected.*

(Optional) Attach Spacers

1. *Attach spacers to the screws of the permanent magnets if you prefer to have legs for the system.*
2. *Fasten securely to ensure stability.*

Final Checks

- *Double-check all connections to ensure they are secure and correctly oriented.*
- *Refer to the GitHub repository for additional visual aids and diagrams if needed.*

Your Maggy Magnetic Levitation Lab System should now be fully assembled and ready for use.

Part III

Software

Introduction

In this part we present the software side of Maggy — all the way from how to program Maggy, to practical implementations of simple controllers and Maggy-to-PC serial communication. This part, together with the next one, is particularly important for students who would like to get started with practical experimentation with Maggy.

This part is divided into three chapters, organized as follows:

Chapter 7 *outlines how to get started with Arduino in order to program Maggy.*

Chapter 8 *details how to program Maggy once the setup is complete. It contains information on how to run tests to ensure that the system is operating as it should, as well as how to implement a simple controller for Maggy.*

Chapter 9 *discusses how to do serial communication between Maggy and a computer, in order to do the control implementations on a computer rather than on the Maggy microcontroller.*

Chapter 7

Getting ready for programming with Arduino

Working with Teensy entails programming sketches using the Arduino Integrated Development Environment (IDE). The IDE contains a text editor for programming, a text console and a toolbar with various programming and library management functions. It is the primary IDE related to programming microcontrollers, from Arduino boards to third-party hardware like the Teensy.

The first step is to download the Arduino IDE from Arduino's website. After installing the IDE, some additional installations must be made to get Arduino IDE to work with Teensy.[1]

1. *In Arduino, go to "File"->"Preferences" and find the "Additional boards manager URLs"-field at the bottom of the window. Copy this link to the field: https://www.pjrc.com/teensy/package_teensy_index.json, if the field already has other entries, separate them with a comma. (Figure 7.1)*

2. *Navigate to the library manager on the left toolbar. Search "Teensy" and install the Teensy package by Paul Stoffregen. (Figure 7.2)*

Now, your Arduino IDE can be used to program sketches for the Teensy board. After writing a sketch, you can check it for errors and verify it by clicking the checkmark in the upper left corner. The icon to the right of the "Verify" button both verifies the sketch and uploads it to the connected microcontroller. To start working on the Maggy, all that remains is to connect the system and install a library for the sensors!

[1] PJRC. Tutorial 1: Software setup, a. URL https://www.pjrc.com/teensy/tutorial.html

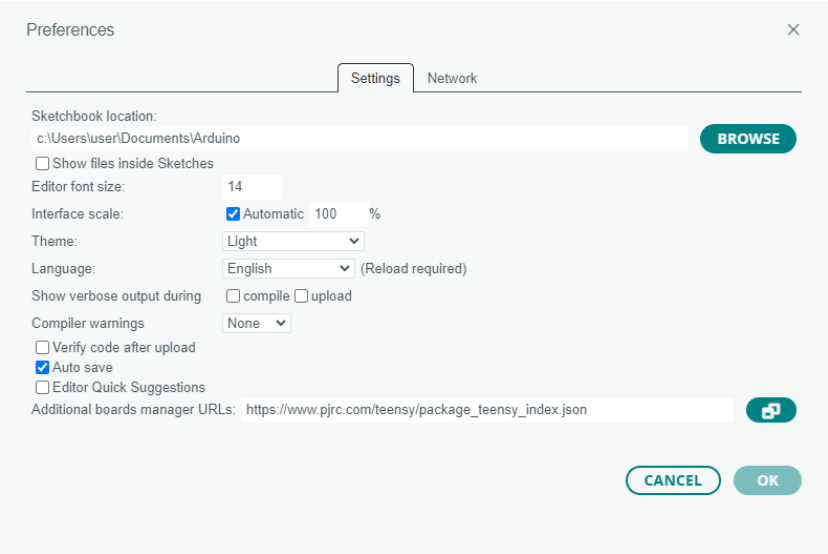


Figure 7.1: Arduino preferences window

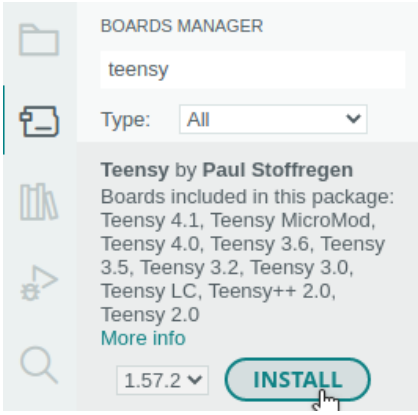


Figure 7.2: Arduino library manager

Chapter 8

System setup & testing

With the proper software in order, you are now ready to connect the system to your computer! The Maggy is connected through the micro-USB port on the Teensy mounted on the underside of the base. Since we want the Maggy to run sketches without being connected to a computer, the power is not supplied from the micro-USB but from an external power supply connected "at the back" of the base. This means that power must always be connected separately to successfully upload your sketches to the Teensy.

In this chapter, we cover the basics related to setup and running the first couple of tests on the Maggy:

1. *Installing libraries*
2. *Connection and configuration in Arduino IDE*
3. *Testing the sensors*
4. *Testing the solenoids*
5. *Running a PD controller*

Installing libraries

Some custom libraries must be installed to successfully utilize all aspects of the Maggy, such as the sensors. These libraries can be found under physical_system/ hardware/ firmware/ libraries in the GitHub repository[1]*. Copy all the folders found here and place them in your own libraries-folder, most often found at C:/ Users/ USER/ Documents/ Arduino/ libraries. The repository is also included as an attachment to this thesis.*

[1] https://github.com/Hansolini/Take-home-Maglev-lab

These libraries are included in sketches using #include <LIBRARY> at the very beginning of the sketch.

Connection and configuration

The next step is connecting the Maggy system to your computer using a USB Type-A to micro-USB cable. The connection itself is pretty self-explanatory, but the configuration in Arduino IDE might not be as straightforward. When the Teensy is connected, it opens a serial port with a given number, for example COM6. This is the port that Arduino IDE must use to communicate with the Maggy. The port is chosen under the "Tools"-menu, where the Teensy will appear under teensy ports in the "Port"-tab as seen in Figure 8.1.

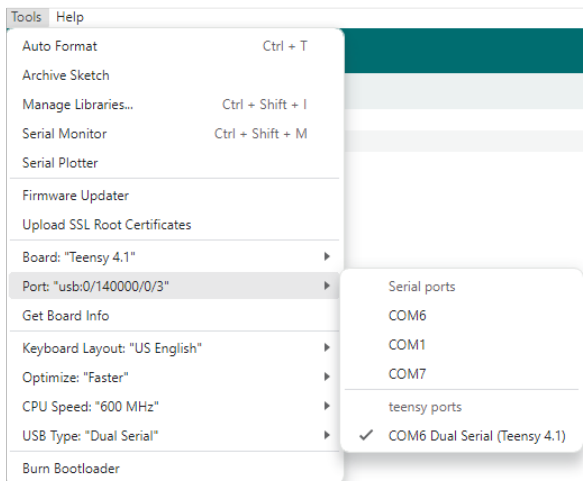


Figure 8.1: The "Port"-menu with Teensy selected

As Figure 8.1 shows, the same menu also contains a tab for board selection. The board must be chosen correctly for Arduino IDE to recognize it. In the case of Maggy, you must select Teensy 4.1 under the "Teensy"-tab.

Sensor test

To assess the quality of the assembly, it makes sense to test the sensors to see if they communicate correctly with the software. A sketch has been developed for precisely this purpose, and is located in the GitHub repository[2] under physical_system/ hardware/ firmware/ system_tests/ solenoid_tests_v2x. The repository is also included as an attachment to this thesis.

[2] https://github.com/Hansolini/Take-home-Maglev-lab

This sketch will start to gather readings from one of the sensors, by default, the sensor with device address A0, which is the sensor in the middle of the base. This data is written to the serial port and can be visualized through

the Serial plotter located in the "Tools"-menu. Sliding the levitating magnet across the base should produce some fluctuations in the magnetic field measurements visible in the serial plotter, as illustrated in Figure 8.2.

Figure 8.2: The Arduino serial plotter

Different sensors can be tested by changing the device address in the sensor object definition. The three sensors have device addresses A0, A1 and A2, as indicated on the board. Listing 8.1 shows the object definition in question, with the device address at the end of the definition.

Listing 8.1: Sensor object definition

```
// Sensors
Tle493d_a2b6 Sensor = Tle493d_a2b6(Tle493d::FASTMODE, Tle493d::TLE493D_A0);
```

Solenoid test

The solenoid test sketch is located in the same system_tests-folder as the sensor test, in the folder solenoid_test_v2x. This sketch runs each solenoid individually for two seconds in a clockwise manner. There are multiple ways to assess the outcome of this test, both through software and physically. The easiest way might be just to hold the levitating magnet above the center of the base and feel it change direction slightly as the solenoids are activated. The change in input signal for each solenoid can also be heard.

Another way is to use the serial plotter, as we did for the sensor tests. You should be able to see small changes in magnetic measurements as each solenoid is activated. However, the changes are quite small relative to the natural noise the sensors produce, so they might not be as easy to spot.

Running a PD controller

Now we are prepared to see the levitating magnet actually levitate! The GitHub repository[3] *contains a sketch in which a PD controller is implemented, it is be found in student_contributions/ code/ PIDKF_master/ General_control/PD as main_maggy_2_with_corrected_input. The repository is also included as an attachment to this thesis.*

[3] https://github.com/Hansolini/Take-home-Maglev-lab

The sketch is pretty much "plug-and-play". After uploading the sketch to the Maggy, take the levitating magnet and carefully place it atop the center of the base. You should be able to feel the solenoids starting to control the magnet. At this point, you can let go of the magnet and watch it levitate. The controller is already tuned, but it can be tuned further by modifying the gain variables Kp and Kd. More details regarding the controller can be found in Chapter 3.

Chapter 9

Serial communication for off-chip computations

A central aspect of the Maggy control is the sending, receiving, and management of data. There are many ways to do this, and the best method is entirely dependent on what kind of data is being transmitted and why it is being transmitted. The most crucial part of data transfer when controlling Maggy is minimizing the time delay between when the measurements are read and when the input is computed. Any delay represents a possible source of instability, and must therefore be kept to a minimum in order to provide the best basis for stability for the system.

The optimal way to do this is, of course, to do all data management and calculations directly on the microcontroller. This presupposes that all code has been developed and compiled in the Arduino C++-variant. If this is not possible or not optimal for the code's purpose, the alternative is to do the computations off-chip through serial communication.

Serial communication

When dealing with data that needs extensive computation, transferring it to a device with more computational power might be a good idea. This can be done through serial communication. Before discussing the specifics, it is useful to describe some core terminology related to this topic.

Serial port *An interface through which data can be transmitted one bit at a time. Most often through connections between a computer and various peripheral devices. The traditional serial port has been mostly phased out on modern devices.* [1]

USB *Universal Serial Bus is an industry standard for data transfer and power delivery between devices. It has largely replaced serial ports due to its superior speed and user-friendliness. Still, in the case of serial commu-*

[1] Vangie Beal. Definitions: Serial port, 2021. URL https://www.webopedia.com/definitions/serial-port/

nication through USB-to-serial adapters, the operating system installs a driver that creates a virtual serial port. This allows for traditional serial communication through modern USB cables.

COM *Simply a name given to any device connected to a serial port. Any driver connected to a serial port will create a device with COM in its name.*

When handling serial communication through USB, an adapter is needed for compatibility. This is solved using the Universal Asynchronous Receiver-Transmitter (UART) protocol, the most used device-to-device communication protocol. As mentioned in its description, a serial port allows for bit-by-bit data transfer. UART bundles byte-sized sequences of bits (which can be adjusted between 5 and 8 bits) and sends them as complete packets, allowing for more robust transmission of longer bit structures.

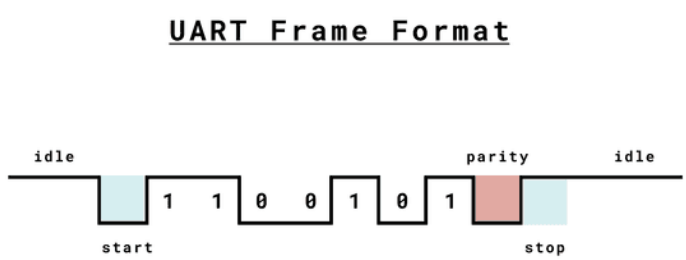


Figure 9.1: Illustration of the bit-allocation of a UART packet

Figure 9.1 shows the bit allocation of a standard UART-packet. Note that the data field does not have a fixed length as shown here, it is dependent on the pre-determined size of the UART-packet.[2]

The start of the UART packet is marked by a start bit, which precedes the actual data bits. The data bits can be followed by a parity bit, which helps detect transmission errors. The packet is concluded by one or more stop bits. The most normal configuration is one stop bit. In addition to this, there are multiple settings for the packets, which must be the same for both sender and receiver, such as data length and data type/encoding.[3]

The UART is an asynchronous protocol, as indicated by its name. This means that it does not rely on a synchronized clock signal but rather on

[2] Hannes Siebeneicher. Universal asynchronous receiver-transmitter (uart), 2023. URL https://docs.arduino.cc/learn/communication/uart/

[3] Hannes Siebeneicher. Universal asynchronous receiver-transmitter (uart), 2023. URL https://docs.arduino.cc/learn/communication/uart/

a defined baud rate. The baud rate determines the speed of transfer, more specifically number of signal changes per second.[4] *Because of this, the baud rate must be the same for both sender and receiver for communication to be successful.*

[4] *Principles of electronic communication systems*. McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 2014

The Teensy 4.1 has a mounted micro USB port to handle communication with other devices, such as a computer. The Teensy can be configured to operate three different virtual serial ports through this connection. This allows for simultaneous communication between multiple interfaces, all at the same high speed. Concerning speed, the Teensy does not abide by the same limitations as, for example, an Arduino UNO board, where the baud rate must be configured to a range between 9600 and 57600. Highly optimized code for USB transfer allows the Teensy 4.1 to operate on native USB speeds, between 12 and 480 Mbit/s. The baud rate setting is therefore redundant when communicating with a Teensy 4.1 microcontroller.[5]

[5] PJRC. Teensyduino: Using usb serial communication, b. URL `https://www.pjrc.com/teensy/td_serial.html`

Measuring data transfer speed

Keeping the delay at a minimum is synonymous with making the data transfer speed as fast as possible. Therefore, it is useful to obtain some measurement of the native speed of the Teensy serial ports. This can be done by running a specialized Arduino sketch developed by one of Teensy's developers, Paul Stoffregen.[6] *The code is also included in the attached zip-folder of the Maggy repository.*

[6] Paul Stoffregen. Usb-serial-print-speed-test. `https://github.com/PaulStoffregen/USB-Serial-Print-Speed-Test/tree/master`, 2019

The logic behind the code is relatively straightforward.

- *A counter keeps track of all loops of the code.*
- *Every second an if-statement updates a parameter called count_per_second, which then gives a measure of how many loops the code goes through every second.*
- *This count_per_second is then printed to the serial port for sending or viewing in the monitor as "lines/sec" along with some other information.*

Through this code, an estimate of the data transfer speed can be obtained. Firstly, the number of characters for each line must be counted, normally each line is comprised of 33 characters. Each character represents one byte which gives 33 bytes per line. From this, the rest of the calculations can be easily deduced.

For example, one running of the code yields the following output in the serial monitor:

```
12:25:38.356 -> count=14850380, lines/sec=544312
```

Note that newline is included as a character in the string above. Therefore, it is comprised of 33 characters instead of the 32 countable ones. From the logic above, this gives the following computations for data transfer speed:

$$\begin{aligned} 544312 \textit{ lines/sec} \cdot 33 \textit{ bytes/line} &= 17962296 \textit{ bytes/sec} \\ &\Downarrow \\ 17962296 \textit{ bytes/sec} \cdot 8 \textit{ Bits/byte} &= 143698368 \textit{ Bits/sec} \\ &\approx 143,7 \textit{ Mbits/sec} \end{aligned}$$

From these calculations the expected data transfer time will be $\frac{1}{143,7 \cdot 10^6} = 6,96 \cdot 10^{-9}$ *sec/Bit, or* $6,96 \cdot 10^{-9} \cdot 8 = 5,6 \cdot 10^{-6}$ *sec/byte. Depending on how many bytes must be sent per transfer, this gives a fairly high frequency. From this, it can be concluded, not surprisingly, that the Teensy is fast enough to support high sampling rates, at least concerning transfer time alone.*

Part IV

Control

Introduction

This part deals with the central aspects related to modeling and control of the system. It provides a theoretical and methodological fundament for experimentation with different aspects of Maggy. Lastly, it deals with controller development and implementation for stable running on Maggy.

This part is divided into six chapters, organized as follows:

Chapter 10 *provides a first-principles model of Maggy, capable of real-time simulation, as well as a description of a Matlab implementation of this model.*

Chapter 11 *presents methods for black-box system identification of Maggy from closed-loop sample data. The section presents several different approaches to this and compares the different methods against each other.*

Chapter 12 *provides a thorough study of the observability of Maggy. This includes the notion of the degree of observability of the system, with comprehensive numerical studies used in the design of optimal sensor placement.*

Chapter 13 *details methods of stabilizing the controller by using both linear controllers and more advanced control methods. The chapter also details the development of a feedback linearizing controller together with an analysis of the stability properties it imposes on the system.*

Chapter 14 *contains studies of two different approaches for implementing a feedback linearizing controller on the system, namely on- and off-chip. Specifically, we evaluate their suitability with regard to system stability.*

Chapter 15 *gives a brief conclusion of the methods and results presented in the two preceding chapters.*

Chapter 10

Physics-driven modeling

Physics-based modeling is a fundamental approach in engineering and science where systems are represented and analyzed using mathematical expressions derived from first principles or physical laws [1]*. This type of modeling is crucial for understanding and predicting.*
Key features of physics-based modeling:

1. *Physics-based models are built on first principles—such as the laws of motion, thermodynamics, or conservation laws—that are universally accepted as true and do not require further validation. These principles provide a solid foundation for developing reliable and predictive models of complex systems.*

2. *The behavior of systems in physics-based modeling is described through mathematical equations. These equations can be algebraic, representing static systems; differential, for systems changing over time; or partial differential.*

3. *One of the most significant advantages of physics-based modeling is its predictive capability. Because these models are derived from fundamental laws, they can predict the behavior of systems under conditions that have not yet been observed empirically. This is particularly valuable in scenarios where experimental testing is expensive, dangerous, or impractical.*

Challenges in Physics-Based Modeling Despite its strengths, physics-based modeling faces several challenges:

1. *Accurately representing a system with mathematical equations can become highly complex, especially when dealing with non-linear behaviors or interactions between multiple factors.*

2. *Solving complex differential equations, especially partial differential equations, can be computationally intensive, requiring significant resources and numerical methods.*

[1] Maki K Habib and Samuel A Ayankoso. Data-driven modeling: Concept, techniques, challenges and future prospects

3. *Many physical systems are highly sensitive to initial conditions and parameter values, which can make accurate predictions difficult if these are not well-characterized.*

The formulation of the system dynamics are detailed in Engmark and Hoang, 2023.[2] *Using Newton-Euler Equations of Motion the system can be modeled for simulation. The mathematical formulation of the system and its dynamics can then be expressed in a state space with a state vector*

$$\boldsymbol{x} = \begin{bmatrix} x & y & z & \alpha & \beta & \gamma & \dot{x} & \dot{y} & \dot{z} & \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix}^T .$$

This yields

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B} f(\boldsymbol{x}, \boldsymbol{u}) \tag{10.1}$$

where

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{0_6} & \boldsymbol{I_6} \\ \boldsymbol{0_6} & \boldsymbol{0_6} \end{bmatrix} \tag{10.2}$$

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{0_6} \\ \boldsymbol{I_6} \end{bmatrix} \tag{10.3}$$

and the nonlinear term

$$f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} m\boldsymbol{I_3} & \boldsymbol{0_3} \\ \boldsymbol{0_3} & \mathcal{I} \end{bmatrix}^{-1} \left(\begin{bmatrix} \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{u}) \\ \boldsymbol{\tau}(\boldsymbol{x}, \boldsymbol{u}) \end{bmatrix} - \begin{bmatrix} \boldsymbol{F_g} \\ \boldsymbol{\omega} \times \mathcal{I}\boldsymbol{\omega} \end{bmatrix} \right) \tag{10.4}$$

where force $\boldsymbol{F}$ *and torque* $\boldsymbol{T}$ *are the resulting cross products of the magnetic field and the magnetization surface current vector* $\boldsymbol{K}$ *and the magnetic polarization vector* $\boldsymbol{J}$ *respectively. (sec.* **??***)*

To limit computational complexity these calculations can be approximated as a summation over n partitions of the magnet

$$\boldsymbol{F} = \frac{2\pi R l}{n} \sum_{i=1}^{n} \boldsymbol{K}(\alpha, \beta, \gamma) \times \boldsymbol{B}_{base} \ (x_i, y_i, z_i) \tag{10.5a}$$

$$\boldsymbol{\tau} = \frac{2\pi R l}{n\mu_0} \sum_{i=1}^{n} \boldsymbol{J}(\alpha, \beta, \gamma) \times \boldsymbol{B}_{base} \ (x_i, y_i, z_i) \, . \tag{10.5b}$$

$\boldsymbol{B}_{base}$ *represents the magnetic field present in the point* (x, y, z) *as produced by the* n_m *permanent magnets and the* n_s *solenoids with center points in* $(x_i, y_i.z_i)$ *in the maglev-base, and is defined as*

$$\boldsymbol{B}_{base}(x, y, z) := \sum_{i=1}^{n_s+n_m} \boldsymbol{B}(x - x_i, y - y_i, z - z_i), \tag{10.6}$$

$$\boldsymbol{B}(x, y, z) = \begin{bmatrix} B_\rho cos B_\phi \\ B_\rho sin B_\phi \\ B_z \end{bmatrix} . \tag{10.7}$$

[2] Hans Alvar Engmark and Kiet Tuan Hoang. Modeling and control of a magnetic levitation platform. *IFAC-PapersOnLine*, 56(2):7276–7281, 2023. ISSN 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2023.10.338. URL https://www.sciencedirect.com/science/article/pii/S240589632300705X. 22nd IFAC World Congress

The total magnetic field from the base is calculated using the formula for magnetic force generated by a thin wire loop in polar coordinates as

$$\boldsymbol{B}(\phi,\rho,z) = \begin{bmatrix} \phi \\ -\frac{z}{\rho}ck\left[\mathrm{K}\left(k^2\right) - \frac{\rho^2+R^2+z^2}{(\rho-R)^2+z^2}\mathrm{E}\left(k^2\right)\right] \\ ck\left[\mathrm{K}\left(k^2\right) - \frac{\rho^2-R^2+z^2}{(\rho-R)^2+z^2}\mathrm{E}\left(k^2\right)\right] \end{bmatrix} \tag{10.8}$$

as described in Chapter 2, where

$$k^2 = \frac{4R\rho}{(R+\rho)^2+z^2}$$
$$c = \frac{\mu_0}{4\pi\sqrt{R\rho}}NI.$$

Chapter 11

Data-driven modeling

In practice, we can observe the input u and the output y of a dynamic system but not the disturbance v. While the input u can be directly manipulated, the output y cannot. Nevertheless, the measured input-output data can still provide valuable insights into to the system's behavior, even without knowledge of the system's internal structure. Thus, mathematical models can be constructed to describe the system dynamics based on observed input-output data [1].

[1] Tohru Katayama et al. *Subspace methods for system identification*, volume 1. Springer, 2005

Dynamic models used for prediction and control can be categorized as either parametric or nonparametric. In parametric modeling, a specific model structure is chosen, and the model is described in terms of a finite set of parameters. Examples of parametric models include transfer functions, state space models, and time-series models. On the other hand, nonparametric methods do not rely on a predetermined model structure, instead, they utilize the system's observed response. Examples of nonparametric models include impulse responses, frequency responses, and spectral density functions. Although parametric methods are more computationally demanding, they produce high-quality models and provide detailed insights into the system. Oppositely, nonparametric methods are generally simpler and require less computational effort [2].

[2] Maki K Habib and Samuel A Ayankoso. Data-driven modeling: Concept, techniques, challenges and future prospects

System identification is a technique primarily used in automatic control that allows for the selection of the optimal model(s) from a specified set based on observed input-output data. The problem of system identification is defined by three key elements:

- *A dataset $\mathbb{D}$ obtained by input-output measurements.*
- *A model set $\mathbb{M}$, or model structure, containing candidate models.*
- *A criterion, or loss function $\mathbb{L}$, to select the best model(s) or a rule to evaluate candidate models based on data.*

The input-output data $\mathbb{D}$ are obtained through experiments. To ensure that the data accurately reflects the characteristics of the system's behavior, it is crucial to design the experiment. This involves selecting which input and output signals to measure, as well as the sampling interval. Useful data for system identification often requires some prior information or physical knowledge about the system. In some cases, open-loop experiments may be impracticable due to safety, technical, or economic constraints. Therefore, data collection often needs to occur under standard operating conditions.

Choosing an appropriate model set $\mathbb{M}$ can be challenging in system identification; however, various classes of discrete-time linear time-invariant (LTI) systems are often used. Since these models do not necessarily reflect the knowledge about the system structure, they are often described as black-box models. One of the most challenging tasks is determining an effective model structure or establishing the orders of these models based on the input-output data provided. One solution to this problem can be solved with the Akaike information criterion (AIC) [3].

[3] Hirotugu Akaike. Canonical correlation analysis of time series and the use of an. 1976

Models incorporating physical principles are termed gray-box models because they use basic physical laws to describe a system or phenomenon, even though some parameters remain unknown. The next step of system identification is to find a model in the model set $\mathbb{M}$ that best explains the experimental data. To achieve, we require a criterion that measures the distance between a model and a real system. This criterion should not only be meaningful of physical meaning but also simple enough to be managed mathematically. A common criterion based on the input u, the real system's output y, and the model output y_M is typically defined as:

$$V_N = \sum_{t=0}^{N-1} l(y(t), y_M(t), u(t))$$

where $l(\cdot)$ is a nonnegative loss function and N is the number of data points.

In theory, with the three key elements in system identification, it is possible to identify the optimal model $M^* \in \mathbb{M}$. This requires:

- *A condition for the existence of a model that minimizes the criterion.*
- *An algorithm for computing models.*
- *A method for model validation.*

Methods for model validation are used to evaluate whether an identified model is an appropriate representation of the system dynamics. Model

validation depends on how the model is utilized, prior knowledge about the system, and the model's fit to actual data. Additionally, if the primary goal is to design a control system, the control performance of a system configured using the identified model must be evaluated. If the performance is unsatisfactory, it is essential to revisiting earlier phases of system identification, such as selecting the model structure selection or designing experiments.

Models obtained through system identification are valid under some specified conditions, like within a certain neighborhood of a working point, and they do not offer physical insight into the system because the model parameters do not have physical significance. Engineering skills and a deep understanding of the system, reflected as prior knowledge, are essential for developing mathematical models from poorly conditioned data.

It is widely recognized that real systems are often nonlinear, time-varying, and may include delays, with some crucial variables or signals that are unmeasurable. However, it is acknowledged that LTI systems represent the simplest and most significant class of dynamic systems utilized both in practical applications and in academic literature.

Classical Identification methods

Let's consider a system where the true output $y(t)$ is modeled by the equation:

$$y(t) = P_0(z)u(t) + v_0(t)$$

where $P_0(z)$ represents the true plant, and $v_0(t)$ is the output disturbance which includes unmeasurable disturbances, modeling errors, and other forms of noise. Consider the task of fitting a stochastic single-input single-output (SISO) linear time-invariant (LTI) model to a specific set of input-output data. The model is represented as:

$$y(t) = P(z,\theta)u(t) + H(z,\theta)e(t)$$

where $e(t)$ is white noise characterized by a zero mean and a variance of σ^2. In this model, θ in $\mathbb{R}^d$ contains all unknown parameters of the model, excluding the noise variance.

The transfer function of the plant in the model is defined as:

$$P(z,\theta) = \frac{B(z,\theta)}{A(z,\theta)} = \frac{b_1 z^{-1} + \cdots + b_m z^{-m}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}}, \quad n \geq m$$

Note that if the plant includes a delay, some parameters like $b_1, \ldots, b_l$ (where $l \geq 1$) may be zero. We can also define the transfer function of the noise model as:

$$H(z,\theta) = \frac{C(z,\theta)}{D(z,\theta)} = \frac{1 + c_1 z^{-1} + \cdots + c_p z^{-p}}{1 + d_1 z^{-1} + \cdots + d_q z^{-q}}$$

where $H(z,\theta)$ is of minimal phase and satisfy $H(\infty,\theta) = 1$. Define a sequence of observed input-output data up to time $t-1$ as:

$$Z^{t-1} := \{u(k),\, y(k),\, k = 0, 1, \ldots, t-1\}$$

then, the one-step-head prediction of the output $y(t)$ based on the provided input-output data Z^{t-1} can be formulated as:

$$\hat{y}(t,\theta) = H^{-1}(z,\theta)P(z,\theta)u(t) + \left[1 - H^{-1}(z,\theta)\right] y(t)$$

Furthermore, the corresponding one-step prediction error can be expressed as:

$$\epsilon(t,\theta) = y(t) - \hat{y}(t,\theta)$$

thus:

$$\epsilon(t,\theta) = H^{-1}(z,\theta)\left[P_0(z) - P(z,\theta)\right] u(t) + H^{-1}(z,\theta)v_0(t)$$

Given a dataset Z^{N-1}, specifying a particular value for the parameter θ enables the generation of a sequence of prediction errors $\epsilon(t,\theta)$ from $t = 0$ to $N-1$. Initial conditions for these prediction errors are set from $t = -1$ to $-p$. The process of fitting the model to the dataset Z^{N-1} involves selecting $\hat{\theta}$ to minimize the variance of the prediction errors. Therefore, the criterion function is specified as:

$$V_N(\theta) = \frac{1}{N}\sum_{t=0}^{N-1} \epsilon^2(t,\theta)$$

Hence, a category of algorithms designed to minimize a function of prediction errors is typically referred to as the Prediction Error Method (PEM). Since the performance criterion $V_N(\theta)$ is generally a complex function of the system parameters, the problem typically requires iterative descent methods, which can potentially get stuck in local minima.

Prediction Error Method for State space models

The innovation model of a discrete-time LTI system can be formulated as:

$$x_{k+1} = Ax_k + Bu_k + Ke$$
$$y_k = Cx_k + Du_k + e$$

where $y_k \in \mathbb{R}^p$ is the output vector, $u_k \in \mathbb{R}^m$ is the input vector, $x_k \in \mathbb{R}^n$ is the state vector, and $e \in \mathbb{R}^p$ represents the innovation vector, which is assumed to have mean zero and a positive definite covariance matrix R. The matrices A, B, C, D, K are system matrices of appropriate dimensions, and the unknown parameters we aim to estimate are contained in these matrices along with the covariance matrix R of the innovation process.

When applying Prediction Error Methods (PEM) to a multi-input multi-output (MIMO) model, the prediction error $\epsilon(t, \theta)$ is computed using a linear state-space model with inputs $u(t) and y(t)$ of the form:

$$\hat{x}(t+1, \theta) = [A(\theta) - K(\theta)C]\hat{x}(t, \theta) + B(\theta)u(t) + K(\theta)y(t)$$

$$\epsilon(t, \theta) = -C\hat{x}(t, \theta) - D(\theta)u(t) + y(t)$$

where $\hat{x}(0, \theta) = 0$ is the initial condition. Then, the performance index, based on the prediction error $\epsilon(t, \theta)$, used to evaluate the model's fit to the data is defined as:

$$V_N(\theta) = \frac{1}{N} \sum_{t=0}^{N-1} \|\epsilon(t, \theta)\|^2$$

Therefore, the PEM estimates are obtained by minimizing $V_N(\theta)$ with respect to θ, and the covariance matrix R of the innovation process e is estimated by computing the sample covariance matrix of $\epsilon(t)$ for $t = 0, 1, \ldots, N-1$.

Subspace identification

Subspace identification is a method in system identification that constructs an estimate of a sequence of state vectors of the state space model from observed input-output data. This approach is particularly useful in identifying models of linear, time-invariant systems. Below, we provide a detailed explanation of subspace identification, elaborating on the processes, models, and calculations involved.

System Model

A stochastic linear system can be described by a state-space model, which includes equations for both the system state evolution and the output [4]*:*

[4] S Joe Qin. An overview of subspace identification. *Computers & chemical engineering*, 30(10-12):1502–1513, 2006

1. *State Equation:*

$$x_{k+1} = Ax_k + Bu_k + w_k$$

   Here, x_k represents the system state at time k, u_k is the input, and w_k is the state noise.

2. *Output Equation:*

$$y_k = Cx_k + Du_k + v_k$$

   In this equation, y_k denotes the output, and v_k is the output measurement noise.

Kalman Filter and Model Forms

The Kalman filter is employed to estimate the system states if the system is observable. With a known Kalman gain K, the filter updates the state estimates based on the output measurements. This leads to an innovation form of the state-space model, where $e_k = y_k - C\hat{x}_k - Du_k$ (the difference between the actual and estimated outputs) acts as innovation or measurement update:

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + K(y_k - C\hat{x}_k - Du_k)$$

This form is useful because the innovation e_k is treated as white noise, simplifying the analysis and computational processes. We get the following form of equivalent innovation:

$$\hat{x}_{k+1} = A\hat{x}_k + Bu_k + Ke_k$$

The corresponding measurement equation is:

$$y_k = Cx_k + Du_k + e_k$$

Subspace Identification

The core idea behind subspace identification is to derive system matrices (A, B, C, D) and the Kalman gain K by analyzing input-output data. The method uses projections of state vectors, estimated from observed data, onto specific subspaces defined by input and output data matrices.

Data Matrices

Extended matrices for input and output data are constructed as follows:

- *Output Data Matrix $Y_{k|i}$: This matrix groups i subsequent outputs starting from time k, facilitating the capture of output dynamics over time.*

$$Y_{k|i} = \begin{bmatrix} y_k & y_{k+1} & \cdots & y_{k+N-1} \\ y_{k+1} & y_{k+2} & \cdots & y_{k+N} \\ \vdots & \vdots & \ddots & \vdots \\ y_{k+i-1} & y_{k+i} & \cdots & y_{k+i+N-2} \end{bmatrix} \in \mathbb{R}^{p \times N}$$

- *Input Data Matrix $U_{k|i}$: Similarly, this matrix contains i sequential input vectors, starting from time k.*

$$U_{k|i} = \begin{bmatrix} u_k & u_{k+1} & \cdots & u_{k+N-1} \\ u_{k+1} & u_{k+2} & \cdots & u_{k+N} \\ \vdots & \vdots & \ddots & \vdots \\ u_{k+i-1} & u_{k+i} & \cdots & u_{k+i+N-2} \end{bmatrix} \in \mathbb{R}^{km \times N}$$

These matrices help in forming a comprehensive view of the system behavior over a specified time horizon.

System Identification Process

1. *Data Construction: Data matrices, as structured above, are stacked to create a Hankel structure, enabling the analysis of system dynamics across multiple time steps.*
2. *LQ Decomposition: The lower Left Q-orthogonal (LQ) decomposition is commonly used in subspace identification methods to process known data matrices. This decomposition assist in computing certain projection matrices and defining the relevant subspace.*
3. *Projection and Estimation: By projecting future outputs onto a subspace that is spanned by past data and is orthogonal to future inputs, an effective state sequence estimation is derived, leading to the calculation of the observability matrix and system matrices.*

Algorithms and Forms

Different subspace identification methods, like N4SID and MOESP, utilize these projections in different forms to derive the system matrices. These methods can handle multiple input and output channels, providing robustness across different system types and operational conditions.

Advantages of Subspace Identification

Subspace identification methods are preferred for their numerical algorithms and the ability to handle multi-input, multi-output (MIMO) systems without requiring complex, nonlinear optimization methods. These methods are based on robust linear algebra techniques, such as QR decomposition and singular value decomposition (SVD), ensuring reliable results even for complex systems.

Closed-loop vs open-loop identification

Understanding the differences between open-loop and closed-loop system identification is crucial in control systems engineering, particularly when dealing with the unique challenges presented by each setup. Let's look deeper into these methods, their specific challenges, and their application.

Open-loop system identification

Open-loop systems are characterized by a lack of influence from the output on the input. In terms of system identification, this setup can be simpler because the input data (or control signal) is independent of the output. In open-loop system identification, subspace methods such as N4SID and MOESP are often used without the pre-estimation step.

Advantages: The input signals can be designed without considering the output, allowing for more straightforward design of experiments, such as applying white noise or specific test signals to fully excite the system dynamics. Since the output does not affect the input, there's no feedback to distort the input signal or its relationship with the system output, reducing complexity in the model identification process.

Disadvantages: Using open-loop identification on systems that are naturally unstable without feedback is often risky or impractical, as it might require operating the plant under unsafe conditions.

Limited Realism: Many real-world systems operate under closed-loop conditions for reasons of safety, stability, or efficiency. Models derived from open-loop identification might not capture important dynamics that are only observable under closed-loop operation.

Closed-loop system identification

In closed-loop systems, the output directly influences the input through a feedback loop. This is commonly observed in processes that need to maintain

stability or where specific performance metrics are crucial and must be precisely controlled. In closed-loop system identification, subspace methods are also applicable. One such method is the Deterministic-Stochastic Realization (DSR) approach [5].

[5] David Di Ruscio. Subspace system identification: Theory and applications. *Lecture notes, Telemark University College, Porsgrunn, Norway*, 1997

Advantages: Closed-loop identification is often necessary for systems that cannot be operated safely or stably in an open-loop configuration. By identifying the system under closed-loop conditions, the resulting model is more likely to reflect the true operational behavior of the system under normal conditions.

Disadvantages: The main challenge is the correlation between the input and the unmeasurable noise due to the feedback loop [6]*. This correlation can lead to biased estimates if not appropriately addressed. Strategies to handle this include:*

[6] Geir Werner Nilsen. Topics in open and closed loop subspace system identification: finite data-based methods. 2005

1. *Instrumental Variables: These are used to provide estimates that are less biased by the correlation between the input and the noise.*
2. *Modified Identification Algorithms: Algorithms such as N4SID can be adapted to reduce the effects of feedback by using approaches that account for the feedback structure in the identification process.*

Experimental design

Building on the insight from the previous sections, we will now evaluate both open-loop and closed-loop subspace identification methods applied to our Maggy-system. Our aim is to estimate system matrices using methods like DSR, N4SID, MOESP, and then compare their prediction and closed-loop response with the help of LQR. To achieve this goal, we will follow the following steps:

1. *Load the system parameters*
2. *Setup the system*
3. *Check controllability and observability*
4. *Design the initial controller*
5. *Setup the simulation*
6. *Design the reference signal*
7. *Simulate the system with the initial controller*

8. *Preprocess data*
9. *Estimate the state-space models using subspace methods*
10. *Check eigenvalues and tune the controller*
11. *Predict the responses*
12. *Plot the results*
13. *Check the variance accounted for between the signals*

In the following subsection, we will describe the most important parts of the implementation.

Design the reference signal

The excitation of input signals is crucial in system identification, since in closed-loop system identification, if the reference signal is not sufficiently excited, there may not be enough information available to uniquely identify a model or capture the relevant system dynamics [7]*. To precisely excite a system, the reference signal can be designed as a sum of square wave signals, each incorporating randomized phase shifts. The inclusion of randomness makes the signal less predictable and more representative of real-world scenarios where noise and other uncertainties affect system performance. Examples of these reference signal can take the form of:*

[7] Lennart Ljung. *System identification toolbox: User's guide*. Citeseer, 1995

$$x_r(t) = \frac{a_x}{n}\left(square(\omega_x t) + square(\omega_x t + 2\pi rand(1)) + square(\omega_x t + 2\pi rand(1))\right)$$

$$y_r(t) = \frac{a_y}{n}\left(square(\omega_y t + 2\pi rand(1)) + square(\omega_y t + 2\pi rand(1)) + square(\omega_y t + 2\pi rand(1))\right)$$

Here's a breakdown of each component in the formula:

- ω_i *is the angular frequency of the square waves.*
- $square(\cdot)$ *denotes the square wave function.*
- $rand(1)$ *generates a uniform distributed random number between 0 and 1, used here to introduce a random phase shift between 0 and* 2π *radians to each square wave component.*
- a_i *is the amplitude scaling factor.*
- n *is a normalization factor.*

Simulation of the system

Note, the simulation doesn't contain noise for the simulated system.

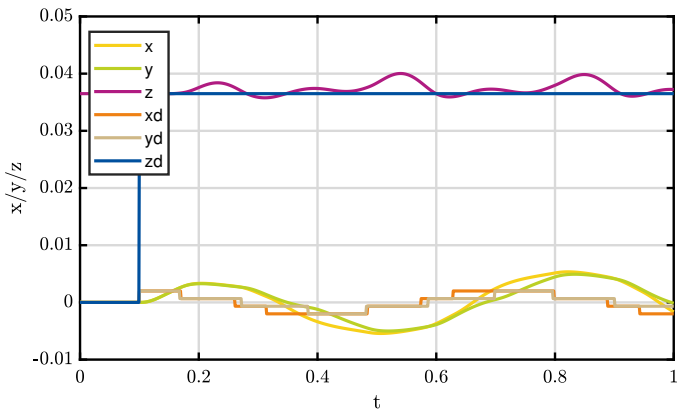


Figure 11.1: Comparison of simulated states against their reference values

Preprocess data

In system identification, "detrending" involves the removal of means, as well as linear or nonlinear drift, from regularly sampled time-domain input-output data [8]*. This process is crucial because it ensures that the data used for modeling accurately represents the underlying system dynamics. This is particularly important because linear models cannot capture arbitrary differences between the input and output signal.*

[8] Handling offsets and trends in data. URL https://www.mathworks.com/help/ident/ug/handling-offsets-and-trends-in-data.html

One-step predictor

A one-step predictor is a model used in time series analysis to forecast the value of a variable at the next time step, based on the current and possibly past values of the series. This type of predictor is designed to provide short-term forecasts, making it particularly useful in applications where immediate predictions are necessary. The primary goal of a one-step predictor is to minimize the prediction error for the next time step, ensuring accurate and reliable short-term forecasts.

The aim is to develop a one-step predictor that incorporates a control strategy using the Linear Quadratic Regulator (LQR) gain to stabilize the closed-loop system. We plan to develop a one-step predictor using system matrices derived from the following methods:

- ***DSR***
- *N4SID*
- ***MOESP***

The one-step predictor is defined by the following equation:

$$y_{k+1} = Ay_k + B(-TsK(y_k - r_k))$$

where:

- *y_{k+1} is the predicted state at the next time step.*
- *Ay_k describes It defines how the state evolves over time.*
- *$B(-TsK(y_k - r_k))$ represents the control input formulated by the deviation from the reference trajectory r_k, with K as the LQR gain matrix designed to stabilize the system.*
- *Ts represents the sample time.*

Real system (derived from physical principles)

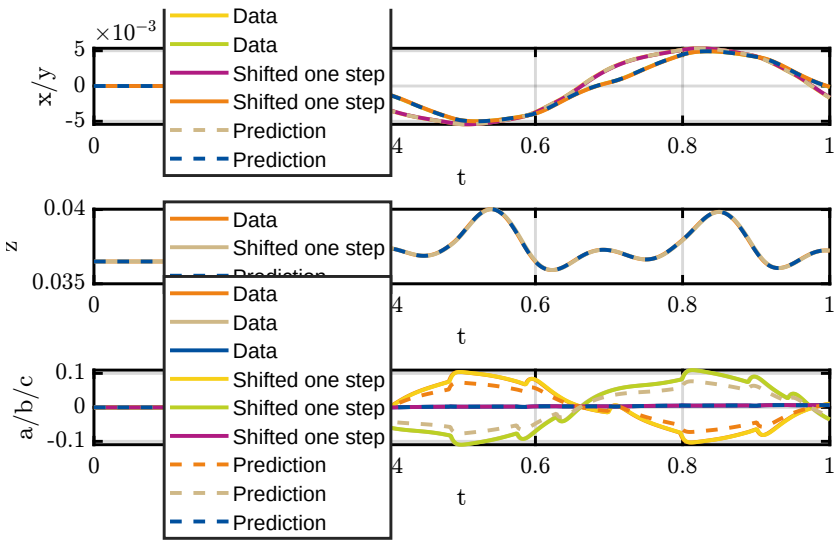


Figure 11.2: Closed-loop response obtained with the real system

DSR

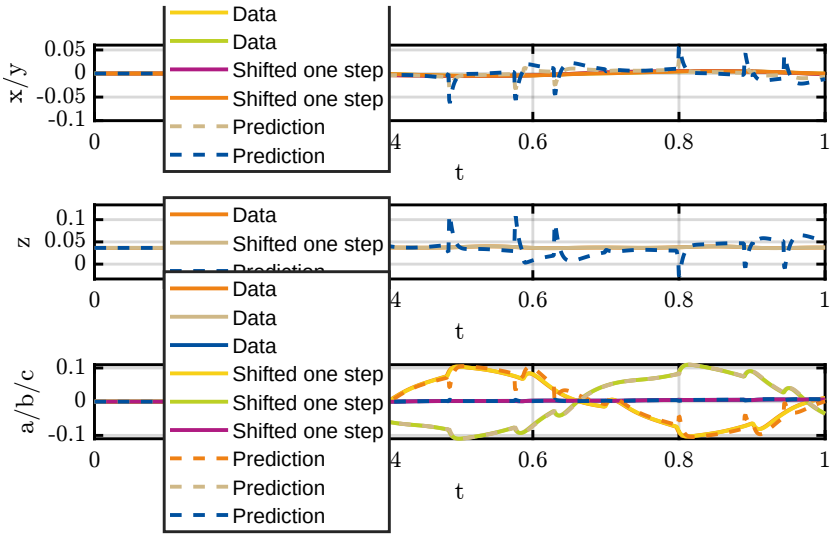


Figure 11.3: Closed-loop response obtained with the DSR method

N4SID

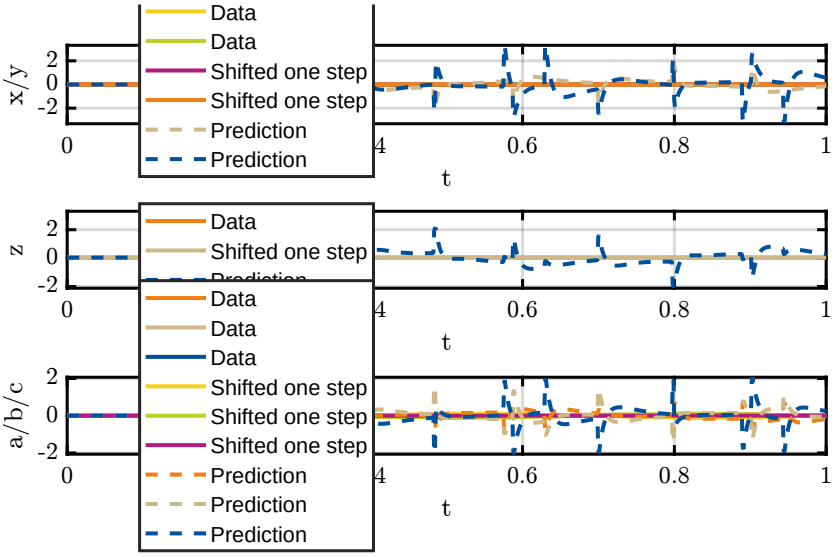


Figure 11.4: Closed-loop response obtained with the *N4SID* method

MOESP

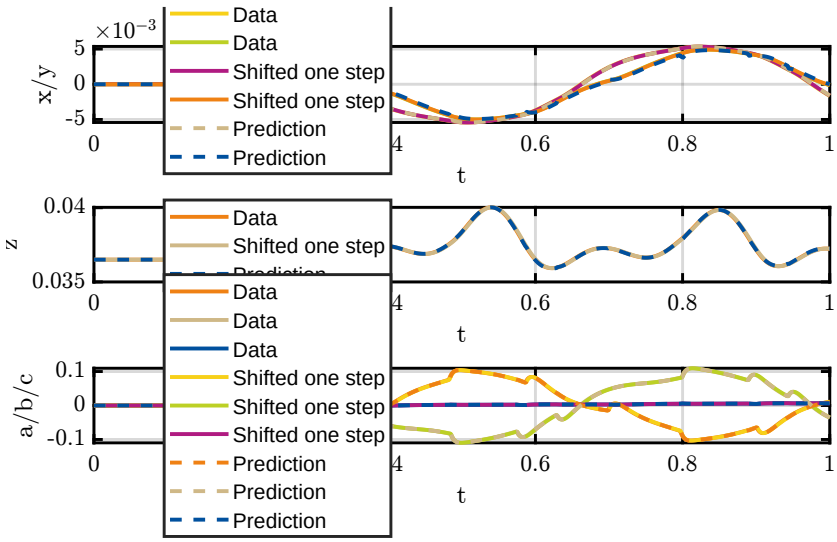


Figure 11.5: Closed-loop response obtained with the *MOESP* method

Variance accounted for

In MATLAB we can use the "vaf.m" function [9] *to verify the correctness of a model, by comparing the real output with the predicted output of the model. If the two signals are the same, then VAF is 100%. If they differ, the VAF will be lower. When y and y_{est} have multiple columns, the VAF is calculated for every column in y and y_{est}.*

[9] Vaf. URL `https://www.dcsc.tudelft.nl/~jwvanwingerden/lti/doc/html/vaf.html`

Real	DSR	N4SID	MOESP
100.00	0.00	0.00	99.96
100.00	0.00	0.00	99.65
100.00	0.00	0.00	99.96
90.84	94.52	0.00	99.97
90.81	99.84	0.00	99.96
99.99	87.76	0.00	99.75
99.13	99.93	0.00	99.97
99.07	99.96	45.33	99.92
99.86	99.84	0.00	99.84
0.00	84.73	72.69	84.72
0.00	84.80	75.56	84.80
68.23	68.19	0.00	68.20

Table 11.1: Variance accounted for between the real output and closed-Loop responses from different estimation methods, given as percentage

Conclusion

Revisiting the closed-loop responses for the different estimation methods and the VAF values given in Table 11.1, it is evident that the MOESP method exhibits better fit with the real data. Additionally, it is noted that the physical matrices that make the "real" state space-model employed for prediction demonstrate a relatively lower fit to the data. Furthermore, the N4SID method exhibits the least accurate predictions, which is also evidenced by the corresponding VAF values.

Tips for furtherwork:

- *The DSR method might benefit from further tuning.*

Chapter 12

State observers

Observability

Observability is an important property of a state-space system. It is a measure of how easy it is to estimate the internal state variables of the system, based on output variable trajectories. In such a system an observer can use the output variables (They will also be called measurements here) of the system to estimate the internal state variables, but only if the system is observable. Observability does not need to be a binary property, meaning that a system can be hard to observe even though it is observable. If that is the case, the quality of the state estimates from an observer might not be good enough to be used for control.

Background, why check observability?

Observer design for Maggy has already been attempted in the specialization project.[1] *This work did not reach its goal of arriving at an observer giving state estimates good enough for use in control. One possible reason for why this work did not succeed, is that the Maggy's model might be close to being unobservable. Therefore, the observability of the model should be analyzed more thoroughly before attempting to continue the work of the specialization project.*

[1] Sverre Graffer. State estimation of a maglev system with the luenberger observer. 2023

A common way to test the observability of a system is by checking the rank of its observability matrix, which is calculated from the system's model. This method has some significant limitations. Firstly, it can only be used directly on linear models. This means that for nonlinear systems, a linear approximation of the system's model has to be used, which might not have the same observability properties as the nonlinear model it is based on. Secondly, this method only gives a binary result when it describes the observability of the system. This means that a system that is very hard, or even close to impossible to observe, might be described as observable by this

method. Because of its limitations, the method described above is not suitable for determining whether Maggy is observable enough for the purpose of control. A different method is therefore needed.

The desired properties for a method are that it should work directly on the nonlinear model and give an indication of how observable the system is, not only if it is observable. A method with these properties will help answer the question, is an attempt at creating an observer for the system worth the effort?

Finding a method that has the properties described above will open up new possibilities. Firstly, it makes it possible to choose two of the state variables that make up the state space and perform the observability check in a grid pattern in a plane spanned by the two selected state variables. The results can then be presented in a heat map, with the two axes of the heat map representing the two selected state variables, making it easy to see how the observability responds to changes in these two states. This method also opens up the possibility of testing observability versus sensor placement and therefore making it possible to optimize the placement of the sensors.

Description of the method

A search for such a method resulted in finding a paper[2] *which describes a suitable method. It provides a method that has the desired characteristics described above. The paper provides a method to calculate two numbers for every possible state in the state space. The authors of the paper call these two numbers "local unobservability index" (LUI) and "local estimation condition number" (LECN). The two numbers say something about how difficult it is to estimate the states of the system when the system is in the state in which the method checks the observability. In short, LUI being a high number means that the estimates will be strongly affected by measurement noise, and LECN being a high number means that the estimation problem is ill-conditioned.*

[2] Arthur J. Krener and Kayo Ide. Measures of unobservability. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6401–6406, 2009. DOI: 10.1109/CDC.2009.5400067

One thing to note about this method is that it only considers models of the type 12.1. In other words, models where the state dynamics and the measurement function are only dependent on the current state.

$$\begin{aligned} \dot{x} &= f(x) \\ y &= h(x) \end{aligned} \tag{12.1}$$

This means that to use this method on a system with inputs, which Maggy's model is, the model to be tested for observability must somehow be reduced to a model without inputs. One simple way to do this is to set all the inputs to constants, for example to zero. A different and more complex

approach would be to insert some controller into the function for the state dynamics such that the input becomes a function of the states. This approach might get hard quickly, especially if the controller requires an observer to function. Both approaches will make it possible to test the system using the method, but it is not obvious how much difference the choice of approach will make on the results of the test.

The method consists of two main steps. The first step is to obtain the empirical local observability Gramian. The second is to compute the numbers, LUI and LECN. To obtain the elements of the empirical local observability Gramian, one can use equation (4) from [3] *which is presented here as* (12.2)*. This formula is used to calculate the element in row i and column j in the empirical local observability Gramian,* $P(x^0)$*:*

[3] Arthur J. Krener and Kayo Ide. Measures of unobservability. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6401–6406, 2009. DOI: 10.1109/CDC.2009.5400067

$$\frac{1}{4\epsilon^2}\int_0^T (y^{+i}(t) - y^{-i}(t))'(y^{+j}(t) - y^{-j}(t))\,dt \tag{12.2}$$

In this formula, $y^{\pm i}(t)$ *are the output trajectories resulting from simulating the system with* $x^0 \pm \epsilon \mathbf{e}^i$ *as the initial value.* $\mathbf{e}^i$ *is a unit column vector of the same size as* x^0*, where the element in row i is equal to* 1*, and the other elements are equal to* 0*.* x^0 *is the state in which one wants to calculate the empirical local observability Gramian. After obtaining the empirical local observability Gramian, one can now obtain the two numbers associated with it, LUI and LECN.*

$$LUI = \frac{1}{\sqrt{\lambda_{min}}} \tag{12.3}$$

$$LECN = \sqrt{\frac{\lambda_{max}}{\lambda_{min}}} \tag{12.4}$$

In 12.3 and 12.4, λ_{min} *and* λ_{max} *are the smallest and largest eigenvalue of* $P(x^0)$ *respectively. These two formulas were described in text in* [4] *and written down as formulas here based on the description.*

[4] Arthur J. Krener and Kayo Ide. Measures of unobservability. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6401–6406, 2009. DOI: 10.1109/CDC.2009.5400067

The formulas above do not take into account that different state variables might need to be estimated with different precision. Sometimes the systems state variables describe different physical quantities, which is the case here. In the case of the Maglev model, $(x, y$ *and* $z)$ *are lengths given in meters, while* $(\alpha, \beta$ *and* $\gamma)$ *are angles given in radians. The angles will in this case usually be described by much higher numbers than the lengths, so it makes sense to demand higher precision from the state variable estimates given in meters, than the ones given in radians.*

The paper[5] *comments on this issue briefly but does not give any detailed instructions on how to take it into account other than stating that the state coordinates x should be properly scaled before computing the empirical*

[5] Arthur J. Krener and Kayo Ide. Measures of unobservability. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6401–6406, 2009. DOI: 10.1109/CDC.2009.5400067

local observability Gramian. One way to interpret this, which is the way it is done here, is to define a new set of state variables, $\tilde{x}$, which are scaled versions of the original state variables. Equation 12.5 is used to convert between the original and the scaled state variables. P_1 is a diagonal matrix where each element on the diagonal has a value such that increasing any state in $\tilde{x}$ by one, increases the corresponding state variable in x by the precision one wants in that state variable from an observer.

$$\tilde{x} = P_1 x \tag{12.5}$$

Instead of running the simulations needed for 12.2 using the original dynamics $\dot{x} = f(x, u)$, a new function based on the original one and the conversion between the states are used instead.

$$\dot{\tilde{x}} = f_{mod}(\tilde{x}) = P_1 f({P_1}^{-1}\tilde{x}) \tag{12.6}$$

When using 12.6, the point x^0 where observability is to be checked needs to be scaled with P_1 such that the initial value for the simulations become $P_1 x^0 \pm \epsilon \mathbf{e}^i$.

The more noise in the measurements, the harder the system becomes to observe. If one has the choice of using two different sensors to measure the same signal, it would make sense that using the sensor with the least amount of noise would lead to the system being more observable than if the other sensor was used. This should be reflected in the number LUI such that using better sensors gives a better score. This means that one has to take into account knowledge about the amount of measurement noise in the sensors that are used.

One way to do this, which is the way that the paper suggests, is to scale all the measurements from the simulations such that the noise covariance matrix of the new measurements is the identity. This can be done in a similar way to the scaling of the state variables, by scaling the output of the measurement function with a diagonal matrix P_2. This means that Equation 12.7 can be used to convert between original and scaled measurements.

$$\tilde{y} = P_2 y \tag{12.7}$$

Before using (12.2) the state trajectories simulated using 12.6 needs to be converted to measurement trajectories. The original measurement function $y = h(x, u)$, needs to be modified a second way in addition to scaling. The original function expects original state variables, x while the state trajectories are scaled $\tilde{x}$ coordinates. The scaling of the states in the state trajectories can be reversed by multiplying the input of the measurement function with ${P_1}^{-1}$. These two modifications leads to formula 12.8.

$$y_- = h_{mod}(x_-) = P_2 h({P_1}^{-1} x_-) \tag{12.8}$$

By using modified versions of both the state dynamics and the measurements presented in 12.6 and 12.8 respectively, the differences in required precision among the state variables and noise levels in the measurements have been taken into account.

To sum up, this has to be performed to check the observability in one state:

1. *The parameters δ, T, $x0$, u, $f(x,u)$ and P_1 must be chosen before continuing to the next step.*

2. *The system is simulated from several slightly different initial values, two simulations for every state in the system.*

3. *Before advancing to the next step, the measurement function $h(x,u)$ and the diagonal matrix P_2 with scaling factors must be decided.*

4. *At this point, measurement trajectories corresponding to all the state trajectories from the last step must be created using the modified measurement function.*

5. *At this point it is still possible to remove one or more measurements from all the measurement trajectories. This might be useful if one wants to check how the observability changes given that some of the measurements in the measurement function are unavailable.*

6. *Now, the empirical local observability Gramian can be calculated based on the (possibly reduced) measurement trajectories.*

7. *States that are not interesting to include in the analysis can be removed at this point by deleting the row and column in the empirical local observability Gramian corresponding to these states. If any states are removed, this matrix will still be square, but its dimension will be reduced.*

8. *In the final step the numbers, LUI and LECN can be calculated from the (possibly reduced) empirical local observability Gramian prepared in the two last steps.*

Note that some parameters have to be chosen early, while others can be chosen later in the process. This means there is potential to reuse some data in some scenarios. For example, when changing the sensor placement, only the measurement function will change which means that state trajectories can be reused while testing different sensor placements. If the simulation process is slow, this approach might save significant time.

Implementation of the method

The model to be tested for observability is implemented in Matlab, so it is convenient to also use Matlab for implementing the method for checking observability. Also, necessary functionality for implementing the method, like simulating dynamical systems and working with matrices are easy to use in Matlab. Therefore Matlab was used to implement the method. It was implemented, as the following Matlab functions:

- *obs_check.m*
- *obs_sim.m*
- *calc_obs_gram.m*
- *lui_and_lecn.m*

The first function (obs_check.m) is the main function, and it is used to test observability in one state. It takes all the necessary parameters and functions as inputs, and outputs LUI, LECN and the state trajectories used during the process. This function does very little on its own and uses the rest of the functions on the list to do the necessary calculations. The state trajectories are returned such that they can be saved and used later.

About the results

The next three sections presents and discusses the results obtained after checking the observability of the model using the method described above. The first section contains pairs of heat maps that show observability as a function of selected state variables of the system. The second section contains plots that show how the observability changes as a function of the placement of the sensors. The third section contains a table that shows the results of removing sensors and ignoring state coordinates, and what effect that has on observability. For all the tests, the model parameters are the same as the ones that are being used in the physical system, Maggy, unless otherwise stated.

The table 12.1 shows what method parameters were used when performing the tests. The reasons behind choosing some of the values in this table are mentioned here. All the inputs in u are set to zero as explained when describing the method. The time T associated with the local observability Gramian should not be too long, since the system is both unstable and fast. The value of ϵ used here is one of the values the authors of the method suggest, and it was also tested and it gave similar results to higher and lower values suggesting that it is low enough without being so low that it gives numerical problems. The scaling factors a and b are not obvious. The numbers used here are the desired precision in an estimate of the associated state coordinates. They were chosen based on intuition. The scaling factors c and

Table 12.1: Method parameters

u	$\begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^\top$
T	0.01
ϵ	0.001
states	x, y, z, α, β
measurements	All measurements
P_1	$\text{diag}\left(\begin{bmatrix} a & a & a & b & b & b & c & c & c & d & d & d \end{bmatrix}\right)^{-1}$
a	0.0005
b	$2\frac{pi}{180}$
c	$100 \cdot a$
d	$100 \cdot b$
P_2	$\text{diag}\left(\begin{bmatrix} \sigma_1 & \sigma_2 & \sigma_3 & \sigma_1 & \sigma_2 & \sigma_3 & \sigma_1 & \sigma_2 & \sigma_3 \end{bmatrix}\right)^{-1}$
σ_1	$\frac{1}{1000}\sqrt{0.0084}$
σ_2	$\frac{1}{1000}\sqrt{0.0085}$
σ_3	$\frac{1}{1000}\sqrt{0.0124}$

d are much harder to choose since they are associated with the time derivatives of the state variables. They were decided purely by intuition. The state variables included are all the non derivative states except γ which is intuitively unobservable and not that interesting anyway. By excluding the time derivative states, the uncertainty of c and d are made less important. Intuitively, if the non-time derivative state variables are observable, obtaining the time derivative states would not be too difficult. The scaling factors σ_1, σ_2 and σ_3 are based on measurements collected from the central sensor in the physical Maggy system. The numbers under the square roots are taken from the diagonal of a covariance matrix calculated using those collected measurements. Since the data was given in mT and not T, a factor is necessary to correct for this. The scaling factors are repeated for each sensor because it is expected that all the sensors have the same noise levels. All the measurements from the three sensors are included because the test should be as relevant as possible to the current model parameters of Maggy.

Heat maps

This first results section shows pairs of heat maps, where each pair is from the same collection of observability tests. All the tests in each collection are equal, except for the state where observability is tested. More specifically, ten of the state variables are kept constant for all the tests while two of the states vary from test to test. The ten state variables that are the same for all the tests in one collection are zero unless otherwise is stated in the caption. Each collection of tests are represented by two heat maps, one representing LUI, and the other LECN. This allows for seeing visually how good or bad the observability is in different parts of the state space. Figure 12.1 is relevant for the three first heat maps (12.2, 12.3 and 12.4). Note that the range of states shown in the heat maps is quite large. Only a smaller area

in the middle of the heat maps corresponds to states where it is reasonable to stabilize the system.

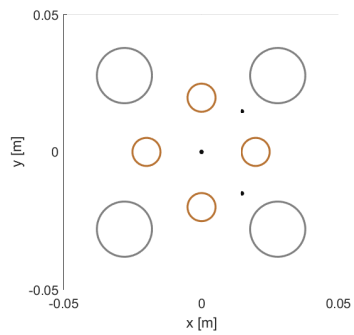


Figure 12.1: x, y (states) displayed as x, y (plot) This plot shows the placement of the permanent magnets, solenoids and sensors.

To makes it easier to interpret the heat maps that follow, 12.1 has been included. It shows a connection between the states in the heat map and the placement of the components.

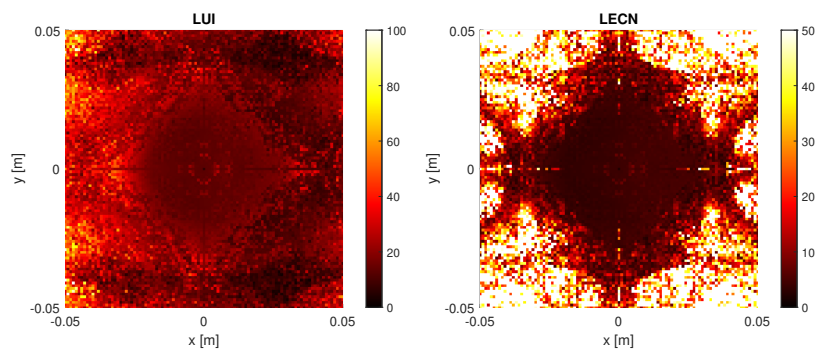


Figure 12.2: x, y (states) displayed as x, y (plot), z = zEq - 0.002 m

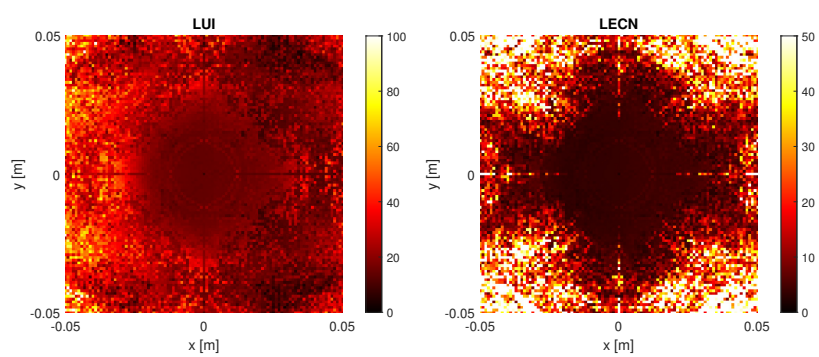


Figure 12.3: x, y (states) displayed as x, y (plot), z = zEq

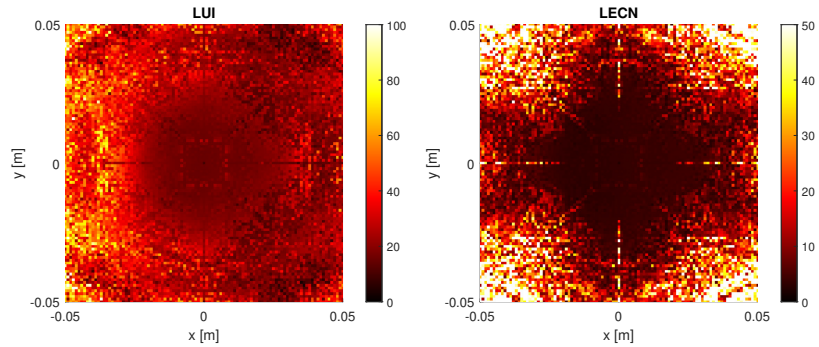


Figure 12.4: x, y (states) displayed as x, y (plot), z = zEq + 0.002 m

The table 12.2 is included to make it easier to compare the first three heat maps. The values in the table are averages calculated from states in four small regions close to the center of the heat maps. A figure that show these four regions are included here 12.6.

heat map	LUI	LECN
12.2 (z = zEq - 0.002 m)	14.5565	4.6190
12.3 (z = zEq)	14.3479	3.9394
12.4 (z = zEq + 0.002 m)	15.2997	3.6517

Table 12.2: Average of region in selected heat maps

In 12.5, states with the angles α, β different from zero are tested for observability.

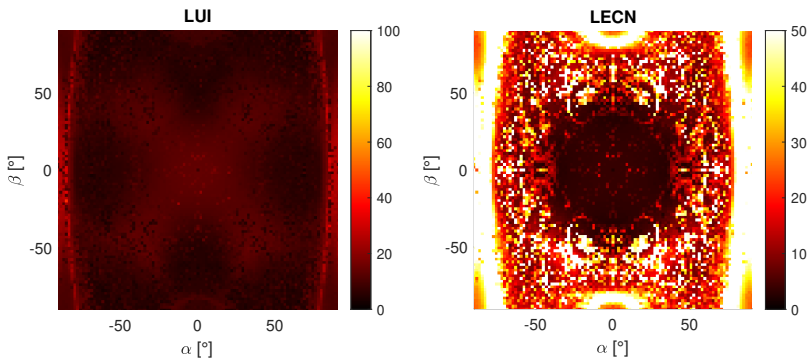
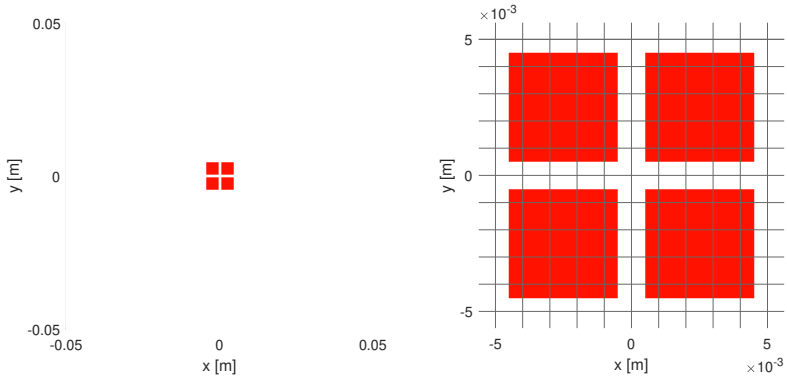


Figure 12.5: α, β (states) displayed as x, y (plot), z = zEq

All the three heat maps with x and y on the two axes,12.2, 12.3 and 12.4, seem to have some properties in common worth mentioning. The heat maps seem to have image noise far out. It does not make much sense intuitively that the actual observability of the system is noisy in this way. One possible explanation is that there might be numerical problems in this area, maybe because the method parameter ϵ might be too high or low for these areas. Another reason might be that the model might not behave well far away from the equilibrium. One can also "see" the permanent magnets in these heat maps, which makes sense because they are an important part of the system dynamics. Another property is that the heat maps are asymmetrical about the y-axis, with better observability when the levitating magnet is close to the sensors, which makes sense. Although, in the small region around the equilibrium, this effect seems to be very small. In fact, all the heat maps seems to have very similar values inside this region, suggesting that this asymmetry is not going to be a problem in practise. When looking at table 12.2 we can see that small differences in height of the levitating magnet only makes small changes to the observability. Interpreting the numbers LUI and LECN with the goal to determine whether the system is practically observable or not is not easy because that would require other examples to compare against. However, intuitively the numbers obtained here does not seem very high, which indicates that the system might be practically observable. One also have to keep in mind that none of the time derivatives state variables corresponding to the first six state variables have been included in these results.

Sensor placement

This section is about creating plots that show observability as a function of different sensor placements. The data shown in these plots will be calculated using the standard sensor placement of Maggy as a baseline. There are several interesting properties of the sensor placement to see the effect of



(a) The states inside the four red areas are being used to make an average of each heat map.

(b) This is a closer view. The intersections between the lines inside the red areas show all the 64 states that are used.

Figure 12.6: The states of interest used for calculating an average. z = zEq. All the other state variables are zero. This is similar to the heat map 12.3.

changing. Four of them have been tested, and the results are shown in this section.

To be able to do this type of test, it is necessary to associate only two numbers, LUI and LECN, to one sensor placement. One way to do this is to test observability in only one state, however, this would not be robust to the case where the results are very different in states close to each other. A more robust approach, that is used here, is to select a number of states close to each other in the most interesting region of the state space and use an average of the results from these states. The 64 states that have been states used for this purpose in these tests are shown in figure 12.6, and they are part of four regions. They were chosen because the state of Maggy is expected to be close to these states while the magnet is levitating. The reason why four regions are used instead of one is that states with exactly $x = 0$ or $y = 0$ or both seemed to often give very different values than the other states in the same region. Another test in the same region with a much finer grid confirmed that only the states with exactly $x = 0$ or $y = 0$ or both stood out this way. Because the state of Maggy will never be in exactly these states while the magnet is levitating anyway, it was decided to remove them.

One obvious property of sensor placement to test the effect of changing is the distance from the z-axis and the off center sensors. This distance is the same as the radius of the circle that these two sensors are placed in. The result from testing this is shown in 12.7. The radius of the circle is shown on the x-axis of the plot. The red dotted line shows what the standard radius for Maggy is.

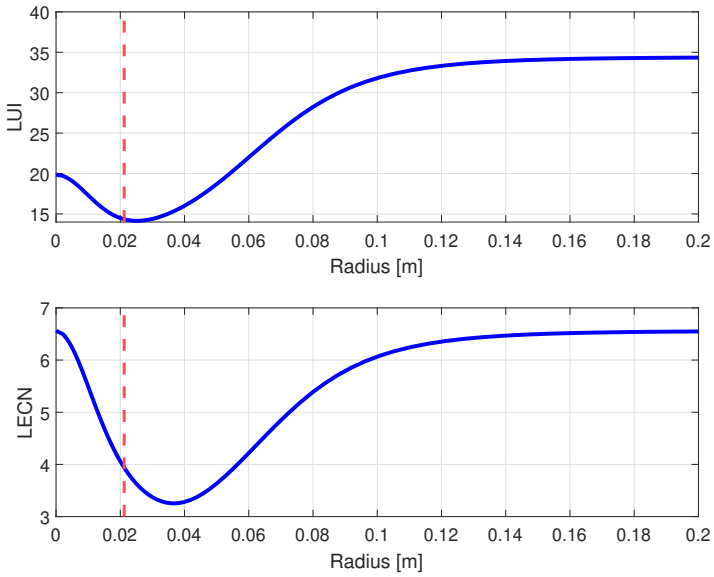


Figure 12.7: Shows observability vs the distance from the two off-center sensors to the z-axis.

Another property that is also interesting is the angle between the sensors and the axes. As standard, the two off-center sensors are placed in a circle in the xy-plane with the center of the circle in origo. The two off center sensors are placed in the first and the fourth quadrant. What we want to do here is to try rotating both the two off-center sensors by the same angle along this circle. Eleven different relative angles spaced evenly with the largest angles being $\pm 45^\circ$ *were tested. These eleven angles were not only tested on the standard radius of the circle. Eleven curves of the same type as 12.7 was calculated for every relative angle. The results are shown in 12.8. A legend showing which curve that corresponds to which angle was not included because all the curves ended up being almost identical, suggesting that the effect of changing the relative angle described here is negligible.*

A third property that it might be interesting to try changing are the height of the sensors. In 12.9, which shows the results from this test, the z-coordinate for all the three sensors are displayed on the x-axis of the plot. The red dotted line shows the standard z-coordinate of the sensors.

The final property that is tested in this section is the angle between the two sensors. The plan for this test is to only change the position of the off-center sensor that is normally placed in the first quadrant. This sensor will be moved along the circle that both the off-center sensors are placed in. A positive angle corresponds to a rotation against the clock, seen from above, while a negative rotation is the opposite. An angle of zero indicates that the

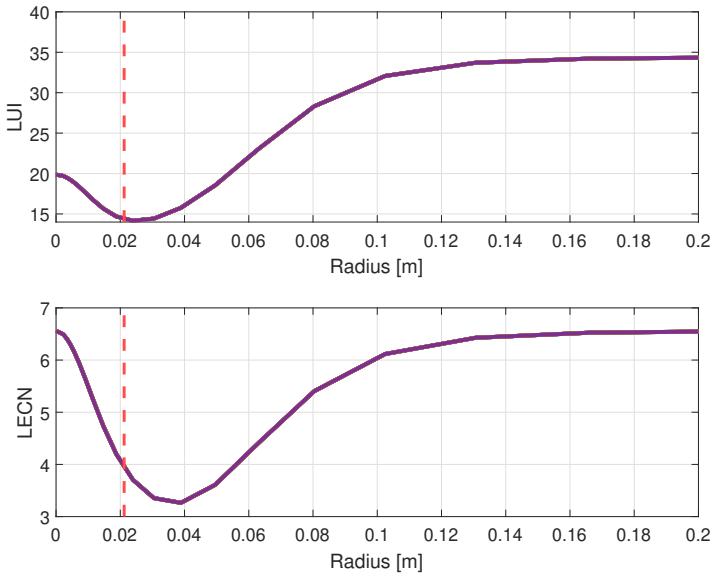


Figure 12.8: Shows observability vs the distance from the two off-center sensors to the z-axis. One curve for every relative angle.

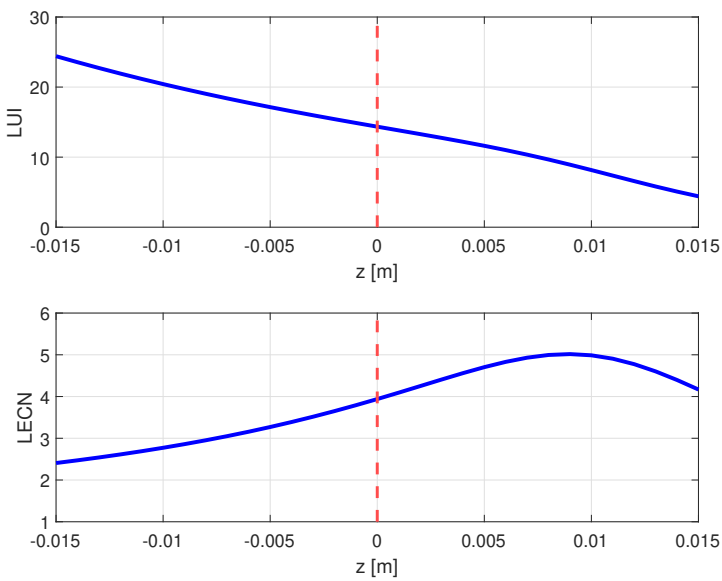


Figure 12.9: Shows observability vs the distance from sensors z coordinates.

sensor has not moved away from its standard position. This means that the angles between the two off-center sensors will vary between 0° *degrees and* 180°.

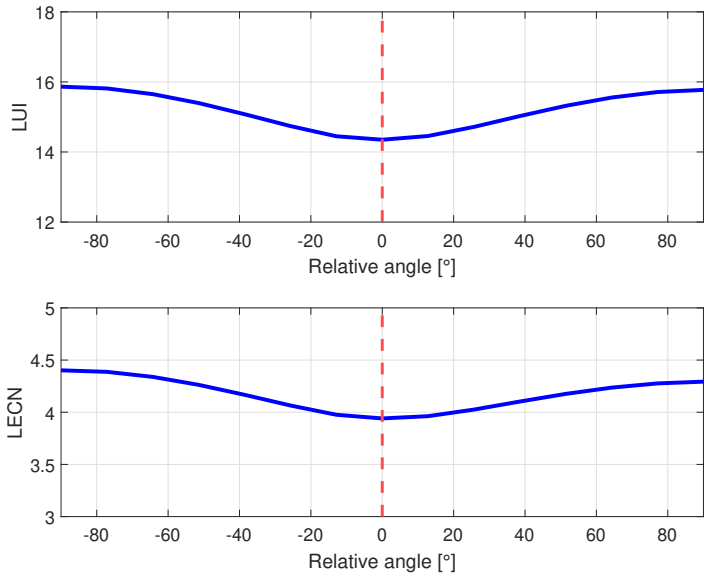


Figure 12.10: Shows observability vs one off center sensor rotated by this angle around the z-axis compared to its normal position.

The plots shown in 12.7 are very interesting. They show that changing the radius of the circle that the off-center sensors are placed in, significantly changes both LUI and LECN. Both plots have an optimal radius and the two optimal radius's are both close to each other and close to the current radius used in Maggy. This means that Maggy already has the off-center sensors placed almost at an optimal distance from the z-axis. It is interesting how both plots have plateaus to the left and right. At the far left of the plots the radius of the circle are zero which effectively means that all the three sensors are in the same central position measuring the same magnetic field. At the far right, the two off-center sensors are starting to get so far away from the levitating magnet that they in practise do not contribute to the observability of the system. This means that far right in the plots describe a situation where there might as well only be one central sensor. This probably explains why LECN is about the same both at left and right in the plot. For LUI, the fact that the right plateau is higher than the left one just means that three sensor at the same point is better than just one. This makes sense because by averaging the measurements from central three sensors, one can obtain measurements with lower noise levels than the same measurements from just one sensor.

It might not be obvious how to interpret the content of 12.8, because it looks very similar to 12.7. The reason why is that it shows 11 different curves that are so similar that they are stacked on top of each other. This means that rotating both the off-center sensors by the same amount around the z-axis makes no difference to the observability of the system. When only

considering the magnetic field contributed by the levitating magnet, it is intuitive that this is the result. However, the observability is also dependent on the system dynamics, and the non-levitating permanent magnets are placed such that this result is not obvious.

The plots shown in 12.9 are quite interesting. Increasing the z-coordinate of the sensors, effectively bringing them closer to the levitating magnet, seems to make the system more observable. This makes sense because the closer the sensors are to levitating magnet, the larger the changes in magnetic field caused by changes in the state variables will be. Even though the effect of lifting the sensors seem to be beneficial, it might also move the sensors into positions so close to the levitating magnet that the sensors are saturated. In that case the sensors will not work at all, so lifting the sensors should not be done without having this potential problem in mind.

The last plot, 12.10 show that the observability of the system changes very little when the angle between the sensors change. Also, the angle between the two off-center sensors on the current version of Maggy is already optimal, being a 90° *angle. This is intuitive because, the other special cases,* 0° *and* 180°*, seems intuitively to be worst placement.* 0° *would mean that the two off-center sensors would be in the same position.* 180° *would mean that all the three sensors would be on a line. The intuition seems to be right because the*

Including different combinations of state variables and output variables

This section tests the effect on observability of including different combinations of states variables and output variables. All the values in these two tables have been obtained with the standard sensor placement for Maggy and the standard method parameters given in table 12.1, except for which states and measurements that are included. The states in 12.6 have been used to calculate average values, just like in the other result-sections. The first table is table 12.3, and the values in this table have all been obtained while using all the measurements from all the three sensors. Observability for many different combinations of states has been tested.

The second table, table 12.4 tests including different combinations of sensors. Sensor one is the central sensor, while sensor two and three are placed off center. Sensor two refer to the sensor in the first quadrant, while sensor three is the sensor in the fourth quadrant. Each sensor gives three measurements, and removing a sensor simply means to pretend like its three measurements do not exist. In this table, six different combinations of sensors are tested, and they are all tested for three different combinations of states.

states	LUI	LECN
x, y, z, α, β	14.3479	3.9394
x, y, z	5.8100	1.5936
α, β	14.1761	1.0622
β, γ	1235.7	89.9
α, β, γ	1391.1	104.1
$x, y, z, \alpha, \beta, \gamma$	1411.4	388.3
$x, y, z, \dot{x}, \dot{y}, \dot{z}$	23.0503	7.1224
$x, y, z, \alpha, \beta, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}$	118.5575	36.674
$\alpha, \beta, \dot{\alpha}, \dot{\beta}$	112.4223	8.4303
$x, y, z, \alpha, \beta, \gamma, \dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}, \dot{\gamma}$	11265	3490
$\dot{x}, \dot{y}, \dot{z}$	10.0997	1.6173
$\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}$	111.2622	17.8348
$\dot{\alpha}, \dot{\beta}$	111.1638	1.1175

Table 12.3: Testing different combinations of different states to include. Measurements from all three sensors are included in this test.

states	x, y, z, α, β		x, y, z		α, β	
Sensors:	LUI	LECN	LUI	LECN	LUI	LECN
1	34.4061	6.5608	8.3578	1.5936	33.8538	1.0790
2	25.6224	3.7566	12.4937	1.8147	25.0742	1.3147
3	25.6442	3.7601	12.4896	1.8143	25.1070	1.3184
1, 2	20.2229	4.7932	6.9040	1.6341	19.9038	1.2118
1, 3	20.2418	4.7979	6.9025	1.6338	19.9287	1.2151
2, 3	15.9324	3.1956	8.4931	1.6970	15.6698	1.0656

Table 12.4: Testing different combinations of different states to include.

From 12.3 we can see that γ is very hard to estimate, probably so hard that it is practically impossible. This is expected, but it is a little bit strange that the number was not even higher. Also the last six state variables (the ones that are the time derivative of the first six) seems to be significantly harder to estimate than the six first, but this is very uncertain because the scaling factors in P_1 corresponding to these states are not hard to chose. The state variables x, y, z and their time derivatives are generally easier to estimate than the angles α, β and their time derivatives. When comparing α, β to x, y, z, α, β we can see that LUI is approximately the same for both while LECN is changing more significantly.

From 12.4 we can see that in the case of using measurements from one sensor, the central sensor works better than either of the two off-center sensors when only x, y, z is included. The opposite seems to be the case when only α, β is included. In that case, using any of the two off-center sensors works better than using the central sensor. It seems that none of the states get dramatically better observability by using two sensors instead of one, and three instead of two. This might suggests that none of the states require two sensors in order to be observable. It is also reasonable to question the usefulness of the measure, LECN. In several cases LECN goes up indicating a less observable system, when the only difference is that additional measure-

ments have been added. Intuitively, it does not make sense that a system can become less observable by adding measurements. However, the increases in LECN caused by adding sensors are quite small.

Conclusion

In this chapter on observability, a method was found in literature and implemented in order to do several tests on the observability of Maggy. Heat maps showing the observability of the system in different states in the state space have been presented. Plots showing the observability of the system as a function of sensor placements have been presented. And tables showing how the observability of the system changes when different combinations of states and measurements have also been presented. A few interesting observations have been found along the way. The most interesting ones are placed in this list:

- *No results here for sure indicates that Maggy is unobservable to the extent that an observer can't be used succesfully on the system.*
- *The observability of the Maggy seems to quite similar in all the states where the levitating magnet will be during levitation.*
- *The standard placement of the sensors seems to be quite good, but there might be potential to increase the observability by placing the sensors higher up.*
- *There is still some uncertainty about the observability of especially the five states $\dot{x}, \dot{y}, \dot{z}, \dot{\alpha}, \dot{\beta}$, because it is hard to decide what is reasonable scaling factors for those state variables.*

This list concludes the part of the text concerning observability.

Classical Linear observers

Future work.

Nonlinear observers

Future work.

Chapter 13

Stabilizing the system

The goal of the Maggy maglev system might not be surprising - to keep a magnet afloat. In order for Maggy to be a productive system for students to experiment on, we must develop some controller that can stabilize the system. In this chapter, we will investigate the system's own stability properties, different controller schemes to use on the system, and the ability of these controllers to actually stabilize it. The methods and results presented here can be made into interactive and challenging assignments for students, ranging from performing a system decomposition to deduce an advanced controller to just implementing a simple PD control scheme.

Earnshaw's theorem clearly states that the levitation of a free magnet in a static, permanent magnetic field without external forces is impossible.[1] *An external force is therefore introduced in the system in the form of the solenoids mounted on the inner circumference of the base. These are used for control through the Teensy, which supplies the solenoids with current, generating the magnetic fields. Therefore, the controller's main job is to manipulate the output current to produce changes in the magnetic field that stabilize the levitating magnet.*

[1] Roberto Bassani. Earnshaw (1805–1888) and passive magnetic levitation. *Meccanica*, 41:375–389, 2006

The nature of Maggy offers a unique opportunity for different control approaches. This is due to its complex attributes like nonlinearity, instability, and underactuation. Despite all this, it is relatively easy to stabilize with basic controllers like a proportional-derivative controller. A state observer will allow for the use of state feedback model-based controllers, such as a linear quadratic controller, or nonlinear variants, such as feedback linearization.

Some introduction of the structure of the chapter?

Controllability

Before a controller can be developed, the system must first be examined to see if it can be controlled at all. This system property is called controllability,

and it describes the ability of the inputs to influence the states and guide them toward a target. For Maggy, this means uncovering which states can be influenced by direct usage of the solenoids and which ones cannot.

The traditional measure of controllability for linear systems is using the controllability Gramian or matrix to assess its controllability. This is more often linked to the output controllability of a system, meaning the ability to alter the measurements gathered from said system by using the input. Since the measurements are often directly linked to the states, this is in most cases directly equivalent to state controllability.

For nonlinear systems, there are several different methods for investigating controllability. The easiest way to prove controllability is to examine the controllability matrix for a linearized system version. This provides a relatively good basis for further investigation. Still, it might not paint a complete picture of the controllability, just like in the case of the observability matrix in chapter 12. Another way to use the controllability matrix to assess the controllability in more detail is to linearize the system in several operating points and then check the controllability matrix.

Maggy is described by a state space with twelve different states. The first six describe the current position of the levitating magnet in the space with its rotation around the three spatial axes, while the last six are the time derivatives of the six former. Of the first six, the three first describe the position in x, y and z coordinates, while the three following states describe the rotations about the x, y and z axes as α, β and γ.

Ringwood, 1995, outlines some interesting properties of the Singular Value Decomposition with regard to matrix rank, that the number of non-singular values in the singular value spectrum can be somewhat equated to the rank of the matrix. Using such a decomposition on the controllability matrix can not only tell us something about the rank of the matrix, i.e., general system controllability but also how controllable the states are in relation to each other.[2]

The GitHub repository[3] *contains a MATLAB script that does this analysis named controllability_and_observability_analysis, located under student_contributions/ code. This code uses MATLAB's integrated function svd() to generate a vector of the singular values and plot them.*[4] *The repo is also included as an attachment to this thesis.*

Figure 13.1 shows the resulting bar plot from the singular value decomposition. Note that it does not explicitly state which states each bar represents. This clearly shows a division of the twelve states into three distinct portions:

[2] John Ringwood. Multivariable control using the singular value decomposition in steel rolling with quantitative robustness assessment. *Control Engineering Practice*, 3(4):495–503, 1995

[3] https://github.com/Hansolini/Take-home-Maglev-lab

[4] Inc. The MathWorks. svd, g. URL `https://se.mathworks.com/help/matlab/ref/double.svd.html`

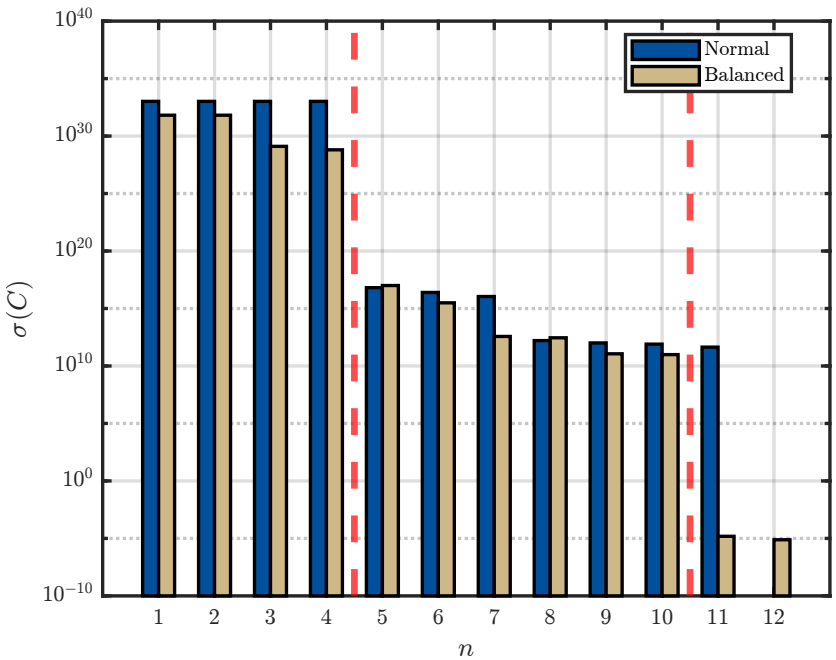


Figure 13.1: Singular values of controllability matrix

one portion with "easily" controlled states, another with slightly harder-to-control states, and a last portion with very hard-to-control states. The plot also includes the results for a balanced realization of the state space. From this, we can conclude that all states have some controllability inherent in them, but we can classify ten of them as controllable and the last two as barely controllable.

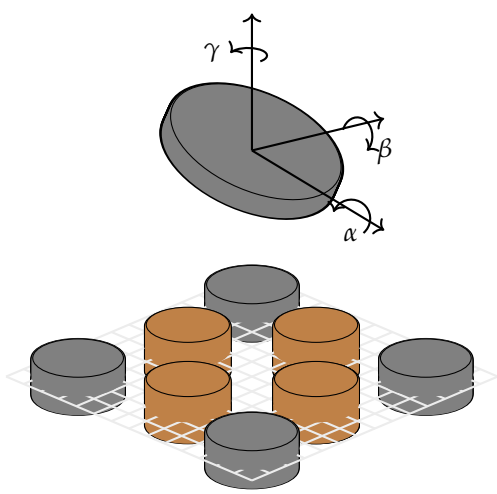


Figure 13.2: An illustration of the system with axes.

For an intuitive description of the controllability of the system, consider the illustration in Figure 13.2. This illustration shows Maggy with its original axes to show the system states we wish to control, both translational and rotational.

It is clear from the figure that translation in the xy-plane is possible by using the solenoids. This also implies that it is possible to apply some torque to the magnet, meaning that the solenoids can directly influence the α and β states, their direction given by the right-hand rule. The solenoids can also maneuver the free magnet along the z-axis, but rotating the magnet around the z-axis is generally considered hard and therefore not controllable. Despite this, there are scenarios in which torque can be applied indirectly by controlling the other angular states.

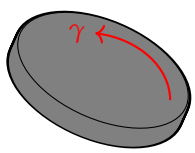


Figure 13.3: An illustration of the levitating magnet in a scenario in which the γ state can be controlled

Figure 13.3 illustrates a scenario where the levitating magnet is tilted along the x- and y-axes. When the magnet is positioned like this, the solenoids can induce some torque around the z-axis, which, in theory, makes the γ state controllable. Despite this, in the context of control, we assume this state to be uncontrollable because it is unobservable. This means that when employing state feedback controllers, there are no states to be fed back; hence, the states are not controllable.

General Control

As mentioned in the chapter introduction, the development of more trivial controllers for the system is absolutely possible. This proves the system's initial controllability and versatility. Trivial controllers in this case include controllers that are usually applied to more linear systems and are generally not considered advanced. Examples of this are PID and LQR controllers, which will be presented in the following sections.

PD controller

A type of controller derived from the PID controller detailed in Chapter 3 is the PD controller. This is a special case of the PID controller where the integral action is removed, and only proportional and derivative action is used. As outlined in Rojas-Moreno and Cuevas-Condor, 2017 the performance of a PID and PD controller for a maglev system is quite similar, note that their

system was comprised of magnets both under and over the free magnet. The PD controller produced a steady-state error of about 6% but an overshoot of just 6% as opposed to 26% for the PID.[5]

[5] Arturo Rojas-Moreno and Cristian Cuevas-Condor. Pd and pid control of a maglev system an experimental comparative study. In *2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*, pages 1–4, 2017. DOI: 10.1109/INTERCON.2017.8079678

With this in mind, a PD controller for Maggy was chosen to reduce potential oscillations due to overshoot to a minimum. The measurements from the hall-effect sensors are adapted to compensate for any bias from the electromagnets and the solenoids. A known issue with derivative action is that it amplifies higher-frequency noise.[6] *Therefore the raw measurements are filtered using a single-pole low-pass IIR-filter on the form*

[6] Carl Knospe. Pid control. *IEEE Control Systems Magazine*, 26(1):30–31, 2006. DOI: 10.1109/MCS.2006.1580151

$$H(z) = \frac{\alpha}{1-(1-\alpha)z^{-1}} \tag{13.1}$$

This filter is used on both the raw measurements and the differentiated measurements, with filter constants α and α_d, respectively. They are implemented as the recurrence relations

$$b[k] = \alpha b_r[k] + (1-\alpha)b[k-1] \tag{13.2}$$

$$b_d[k] = \alpha_d \frac{b[k]-b[k-1]}{\Delta t} + (1-\alpha_d)b_d[k-1] \tag{13.3}$$

where $b[k]$ and $b_d[k]$ are the filtered measurements and raw measurements, respectively, and the subscript d denotes the derivatives of the measurements.

In relation to the controller, the measurements are used directly in the calculation of the error between the reference $r = 0$ and the measurements as

$$\begin{aligned} e =& r - b \\ & \downarrow \\ e =& -b \end{aligned}$$

To simplify the control design, the inputs are redefined using the virtual inputs v_x and v_y, according to

$$\begin{aligned} u_x^+ &:= v_x \\ u_x^- &:= -v_x \\ u_y^+ &:= v_y \\ u_y^- &:= -v_y, \end{aligned}$$

and only two controllers are designed; one for controlling the x-direction (v_x) and one for controlling the y-direction (v_y). The output current provided to the solenoids is sent as PWM-signals from the Teensy. The two solenoids along each axis are supplied with the opposite signals. The logic behind this is as outlined in Brønstad et al., 2022 that when the magnet is deemed to be off-center, the solenoid closest to it will repel it while the opposite one attracts it. This logic largely constitutes the output algorithm.[7]

Pulse-width modulation (PWM): a way of representing an analog signal as a rectangular "wave" with variable time periods.

[7] Martin Brønstad, Jørn Olav Deila, Jørgen Dyrskog, and Markus Langklopp. Magnetic levitation system: Design, prototyping and testing of a digital pid-controller. Bachelor's thesis, Norwegian University of Science and Technology, Trondheim, 2022

The PD controller for Maggy is implemented directly on the Teensy through the Arduino programming language. The controller is discretized for a sampling frequency of $f_s = 5kHz$*, which gives a sample time of* $\Delta t = \frac{1}{5000} = 200\mu s$ *to achieve stable levitation around the system's equilibrium along the z-axis.*

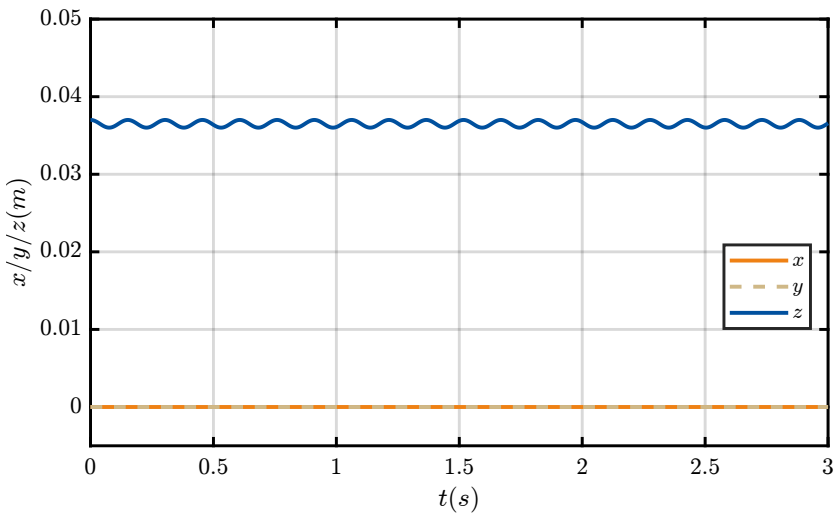


Figure 13.4: Plot of simulation with no initial deviation in the radial direction

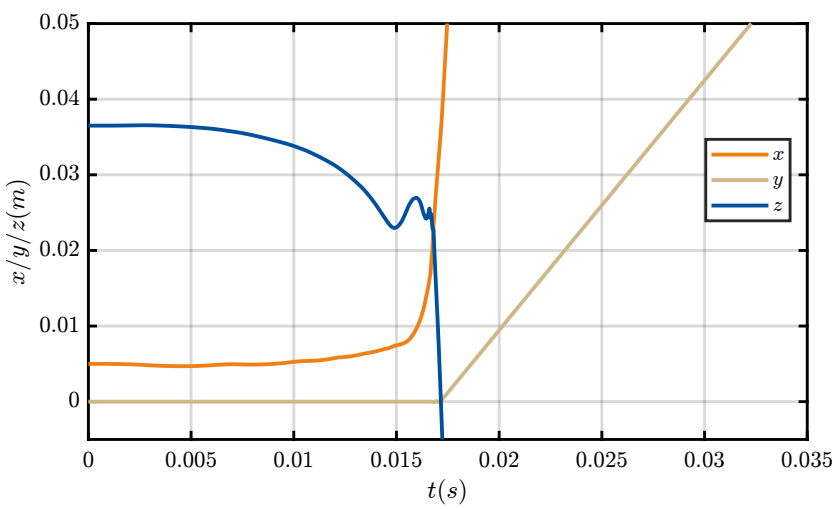


Figure 13.5: Plot of simulation with an initial deviation of 5mm in x direction

Figures 13.4 and 13.5 show the simulated states of the system with the PD controller employed. They clearly show that the system can be controlled by using a PD controller. Simultaneously, it is also evident that the controller leaves much to be desired when it comes to deviations in the radial direction. This is due to the controller not having any knowledge about the

system model, which in turn means that it is entirely unaware of the nonlinearities associated with moving in the radial direction. This is where state feedback model-based controllers might be better suited.

In normal circumstances, the measurements correlate directly to the states of the system, but the nonlinearities of Maggy also manifest themselves in the output. Without some observer or state estimator, a direct deduction of the state of the system is therefore impossible. When the states are directly available, implementing a wide range of model-based state feedback controllers will be possible. This allows for more precise control of the system, as well as more customizable set-point tracking other than just controlling the system to a constant equilibrium.

LQR

The first state feedback controller to be implemented is the LQR controller, as introduced in Chapter 3. This is a linear method, which means that to employ it on the system, it must be linearized around an operating point. This approach has its limitations on a system like Maggy. The main one is that stability can only be guaranteed within a neighborhood of the operating point. Since the system is highly nonlinear, this means that the controller will find it hard to handle disturbances of larger magnitudes.

Despite this, developing such a controller for this system provides a good base for understanding how model-based state feedback controllers are constructed. It also provides a smooth transition from output-feedback controllers to more robust state feedback controllers. In practice, this assumes that a state estimator has been developed and employed on the system, but in the meantime, the controller can be developed using the simulator.

The first step is to linearize the system model. This is usually done around a specific point, called the operating point. This can generally be any point within the operating domain of the system, but in this case, the system's equilibrium along the z-axis is chosen. Just like in the case of the PD-controller, this is done to maximize the potential for stability of the system. Figure 13.6 shows the force balance when moving the magnet along the z-axis in the middle of Maggy's base. The point at which f_z crosses the line is the equilibrium point in the direction of the z-axis and is more specifically $z = 0.0365$. This becomes the operating point of the linearization.

The linearization itself is done numerically due to the complexities posed by the system model. By using finite difference linearization with a step-size of $\Delta = 1e - 6$, the linearized A, B, C and D matrices for the system are found. These matrices can, in turn, be used to develop an LQR controller

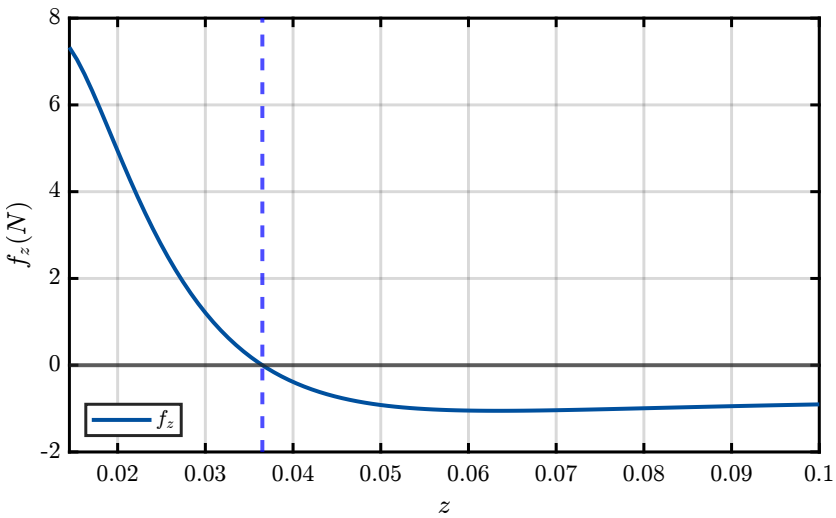


Figure 13.6: Force balance in (x,y)=(0,0) in direction of the z-axis

by solving the algebraic Riccati equation, together with the weight and cost matrices Q and R. This results in a gain matrix K that can be applied in the control law $u = -Kx$ such that the linearized closed-loop system is stable.

In this linearized system model, the Q matrix is designed to strongly favor convergence in only some of the states. Due to the largest nonlinearities being present when the levitating magnet is away from the operating point, the largest weights are placed on x and y to discourage the magnet from straying from the operating point. In the simulation-folder of the GitHub repository[8] a MATLAB live script outlining the LQR process with example code is included. The repo is also included as an attachment to this thesis.

[8] https://github.com/Hansolini/Take-home-Maglev-lab

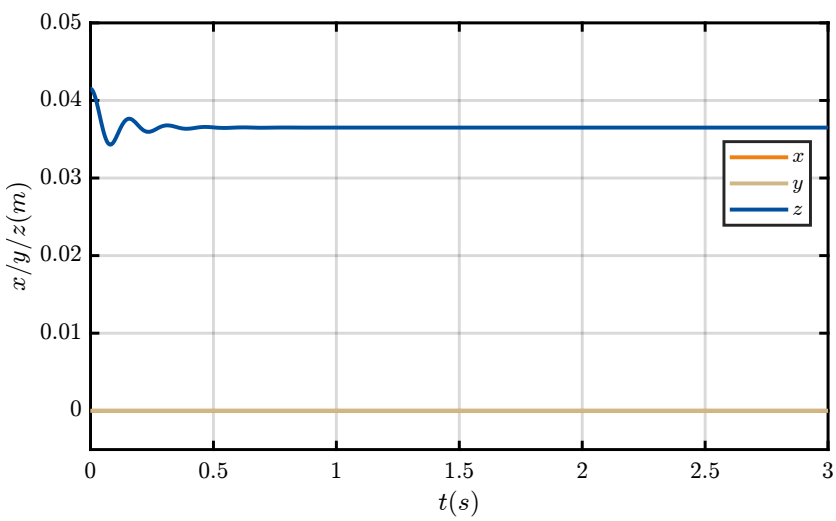


Figure 13.7: Plot of simulation with no initial deviation in the radial direction

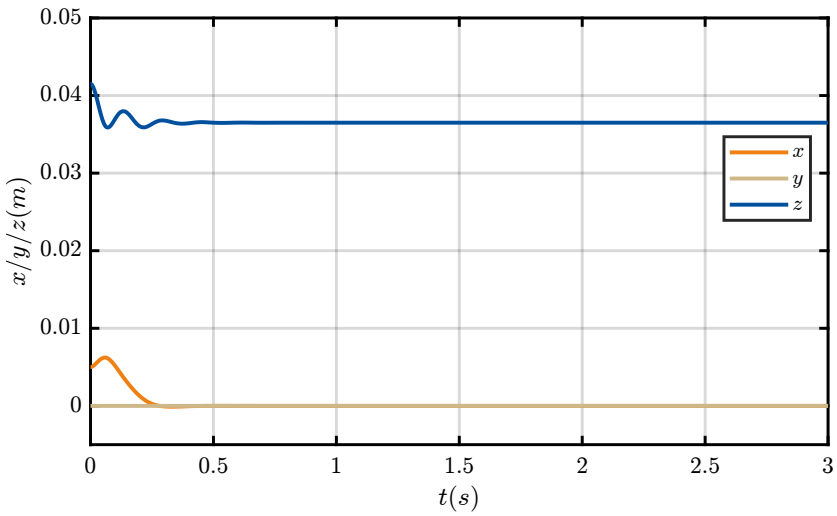


Figure 13.8: Plot of simulation with an initial deviation of 5mm in x direction

Figures 13.7 and 13.8 show the simulation plots for the system with the LQR-controller employed. It is evident that this controller has a better ability to handle deviations in the radial direction than the pure PD controller. It also shows a better ability to converge the states to the reference. Despite this, it is still a linear control scheme, meaning that if the initial point was moved further from the operating point, it would become unstable. Still, this serves as a great example of how transitioning to a model-based state feedback controller will provide a better basis for stability for the system. It also clearly shows the improved tracking abilities.

Now that we have seen how basing our controller on the system model improves stability, it is time to move on to a method more suited to nonlinear systems. We still use the model of the system directly to deduce the controller, but instead of relying on linearized approximations of the dynamics, this control scheme uses exact state transformations to linearize it in "real-time".

Feedback-linearization

The highly nonlinear Maggy system warrants an advanced control method to be stabilized in a larger region. This is achieved by developing a feedback linearization controller scheme for the system. The feedback linearizing controller allows us to retain all the advantages of a model-based controller while guaranteeing stability in a larger region, given that it is able to cancel the nonlinearities directly.

Recall from the presentation of feedback linearization in Chapter 3 that some standard definitions were made to present a system as feedback linearizable, specifically (3.5) and (3.6). To avoid any confusion between the control parameters and the system parameters, we propose the following change in notation:

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bk(\boldsymbol{x})(u - a(\boldsymbol{x})) \tag{13.4}$$

$$u = a(\boldsymbol{x}) + b(\boldsymbol{x})v \tag{13.5}$$

where $b(\boldsymbol{x}) = k^{-1}(\boldsymbol{x})$.

The first step is to check whether the system is input-state linearizable or not. This is done by checking if transformation $\boldsymbol{z} = \boldsymbol{z}(\boldsymbol{x})$ *as described in (3.7) and (3.8).*

In the case of this system, it is evident that by dividing the original state vector into two vectors

$$z_1 = \begin{bmatrix} x \\ y \\ z \\ \alpha \\ \beta \\ \gamma \end{bmatrix}, z_2 = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix}$$

the system can be represented in such a way that all nonlinearities occur in the same state vector, namely z_2. *This is also trivially available from the original form of the system, in which the matrices* A *and* B *already take the form of* A' *and* B'. *From this, it is concluded that the system is feedback linearizable.*

In the specialization project, such a feedback linearizing controller was developed through the steps and method outlined below.[9]

[9] Pål Ivar Delphin Kværnø Fosmo. Control of a magnetic levitation system using feedback linearization. 2023

System decomposition

The system in (10.1) is not input affine, and must therefore be decomposed. By inspecting the formula for the magnetic field $\boldsymbol{B}$ *in (10.8), it is clear that by modifying c, the magnetic field becomes a product with the input I as a linear factor, giving*

$$\boldsymbol{B'}(\phi, \rho, z) = \begin{bmatrix} \phi \\ -\frac{z}{\rho}c'k\left[\mathrm{K}\left(k^2\right) - \frac{\rho^2+R^2+z^2}{(\rho-R)^2+z^2}\mathrm{E}\left(k^2\right)\right] \\ c'k\left[\mathrm{K}\left(k^2\right) - \frac{\rho^2-R^2+z^2}{(\rho-R)^2+z^2}\mathrm{E}\left(k^2\right)\right] \end{bmatrix} I \tag{13.6}$$

where

$$c' = \frac{\mu_0}{4\pi\sqrt{R\rho}}N.$$

Applying this to (10.5a) *and* (10.5b) *yields*

$$\begin{aligned}
\boldsymbol{F}(\boldsymbol{x},\boldsymbol{u}) &= \frac{2\pi Rl}{n}[\sum_{i=1}^{n}\boldsymbol{K}(\alpha,\beta,\gamma)\times\boldsymbol{B}_1'\ldots \\
&\quad \sum_{i=1}^{n}\boldsymbol{K}(\alpha,\beta,\gamma)\times\boldsymbol{B}_{n_m+n_s}']\boldsymbol{I} \\
\boldsymbol{\tau}(\boldsymbol{x},\boldsymbol{u}) &= \frac{2\pi Rl}{n\mu_0}[\sum_{i=1}^{n}\boldsymbol{J}(\alpha,\beta,\gamma)\times\boldsymbol{B}_1'\ldots \\
&\quad \sum_{i=1}^{n}\boldsymbol{J}(\alpha,\beta,\gamma)\times\boldsymbol{B}_{n_m+n_s}']\boldsymbol{I}.
\end{aligned}$$

This allows for the state space to be reformulated as

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bg^{-1}(\boldsymbol{x})(h(\boldsymbol{x})\boldsymbol{I} - p(\boldsymbol{x})) \tag{13.7}$$

where

$$\begin{aligned}
g(\boldsymbol{x}) &= \begin{bmatrix} m\boldsymbol{I}_3 & \boldsymbol{0}_3 \\ \boldsymbol{0}_3 & \mathcal{I} \end{bmatrix}, \\
h(\boldsymbol{x}) &= \begin{bmatrix} \sum_{i=1}^{n}\boldsymbol{K}(\alpha,\beta,\gamma)\times\boldsymbol{B}_1' & \ldots & \sum_{i=1}^{n}\boldsymbol{K}(\alpha,\beta,\gamma)\times\boldsymbol{B}_{n_m+n_s}' \\ \sum_{i=1}^{n}\boldsymbol{J}(\alpha,\beta,\gamma)\times\boldsymbol{B}_1' & \ldots & \sum_{i=1}^{n}\boldsymbol{J}(\alpha,\beta,\gamma)\times\boldsymbol{B}_{n_m+n_s}' \end{bmatrix}, \\
p(\boldsymbol{x}) &= \begin{bmatrix} \boldsymbol{F}_g \\ \boldsymbol{\omega}\times\mathcal{I}\boldsymbol{\omega} \end{bmatrix}.
\end{aligned}$$

The electrical current I has been extracted out as a linear factor, but it is not equivalent to the input vector. This is because the permanent magnets are modeled as solenoids with a constant equivalent current. This means that to have an accurate decomposition, the vector of currents $\boldsymbol{I}$ *must be divided into a constant and variable part.*

$$\boldsymbol{I} = \begin{bmatrix} \boldsymbol{I}_M \\ \boldsymbol{I}_S \end{bmatrix}$$

The new vectors of electrical current are distinguished by their subscripts, M and S for the permanent magnets and the solenoids, respectively. This requires the matrix $h(\boldsymbol{x})$ *to be divided in the same way, giving the decomposed state space formulation*

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bg^{-1}(\boldsymbol{x})(h_S(\boldsymbol{x})\boldsymbol{I}_S + h_M(\boldsymbol{x})\boldsymbol{I}_M - p(\boldsymbol{x})) \tag{13.8}$$

where

$$h_M(\boldsymbol{x}) = \begin{bmatrix} \sum_{i=1}^{n} \boldsymbol{K}(\alpha,\beta,\gamma) \times \boldsymbol{B}_1' & \dots & \sum_{i=1}^{n} \boldsymbol{K}(\alpha,\beta,\gamma) \times \boldsymbol{B}_{n_m}' \\ \sum_{i=1}^{n} \boldsymbol{J}(\alpha,\beta,\gamma) \times \boldsymbol{B}_1' & \dots & \sum_{i=1}^{n} \boldsymbol{J}(\alpha,\beta,\gamma) \times \boldsymbol{B}_{n_m}' \end{bmatrix},$$

$$h_S(\boldsymbol{x}) = \begin{bmatrix} \sum_{i=1}^{n} \boldsymbol{K}(\alpha,\beta,\gamma) \times \boldsymbol{B}_1' & \dots & \sum_{i=1}^{n} \boldsymbol{K}(\alpha,\beta,\gamma) \times \boldsymbol{B}_{n_s}' \\ \sum_{i=1}^{n} \boldsymbol{J}(\alpha,\beta,\gamma) \times \boldsymbol{B}_1' & \dots & \sum_{i=1}^{n} \boldsymbol{J}(\alpha,\beta,\gamma) \times \boldsymbol{B}_{n_s}' \end{bmatrix}.$$

The controller

With the system now decomposed into this form, an input signal to mitigate the nonlinearities can be found by applying the theory directly, and from (13.4) the following definitions can be made

$$k(\boldsymbol{x}) = g^{-1}(\boldsymbol{x}), \tag{13.9}$$

$$a(\boldsymbol{x}) = p(\boldsymbol{x}) - h_M(\boldsymbol{x})\boldsymbol{I}_M, \tag{13.10}$$

which yields

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bk(\boldsymbol{x})[h_S(\boldsymbol{x})\boldsymbol{I}_S - a(\boldsymbol{x})]. \tag{13.11}$$

By comparing (13.11) to (13.4), the corresponding input signal can be found as

$$u = a(\boldsymbol{x}) + b(\boldsymbol{x})v$$
$$\downarrow$$

$$\boldsymbol{I}_S = h_S^{-1}(\boldsymbol{x})(a(\boldsymbol{x}) - k^{-1}(\boldsymbol{x})v). \tag{13.12}$$

What complicates matters here is the presence of $h_S(\boldsymbol{x})$, which must be invertible for all values of $\boldsymbol{x}$. This is the case when the magnet is hovering above the base in an area where the solenoids can control it. Therefore, the matrix is always considered invertible in this case.

With the system decomposed to this form, we are free to choose a control law for the system to implement through v. Multiple controller candidates can be implemented, depending on which properties are desired from the controller. Zhang and Li, 2011, describe the implementation of a PID controller on an electro-hydrostatic actuator, while Uswarman et al., 2013, propose a pole placement scheme for a magnetic levitation system. Given the different nature of these systems, we are inclined to liken our controller to the system that most resembles our own. Therefore, drawing inspiration from Uswarman et al., 2013, the feedback linearizing controller will employ a standard and linear pole placement controller to control the system.[10,11]

[10] Qian Zhang and Bingqiang Li. Feedback linearization pid control for electro-hydrostatic actuator. In *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)*, pages 358–361, 2011. DOI: 10.1109/AIMSEC.2011.6010249

[11] Rudi Uswarman, Adha I. Cahyadi, and Oyas Wahyunggoro. Control of a magnetic levitation system us-

Using MATLAB's place() function, a gain matrix K is generated to place the closed-loop poles of the system in the left half-plane.[12] *To employ this controller to the feedback linearization, the desired input is designed as* $v = -K\boldsymbol{x}$*, giving the final controller*

$$\boldsymbol{I}_S = h_S(\boldsymbol{x})^{-1}(a(\boldsymbol{x}) - k^{-1}(\boldsymbol{x})K\boldsymbol{x}). \tag{13.13}$$

This provides the contours of the chapter's end goal. This entails developing a controller implementation and making adaptations to retain its stability when the hardware's limitations are imposed on it.

[12] Inc. The MathWorks. place, d. URL https://se.mathworks.com/help/control/ref/place.html

State estimator considerations

As discussed in the controllability section, two states in the state space are deemed uncontrollable, namely γ *and* $\dot{\gamma}$*. With the Hall-effect sensors, there is no way for any state observer to deduce the angular rotation around the z-axis of the levitating magnet, let alone tell anything from the measurements. This means that even if these two states were deemed controllable, a state feedback controller would not be able to control them because there are no states to feed back, they are unobservable. The control scheme outlined above is a full state feedback-controller, meaning that all states are controlled. This can be done in the simulator but not in practice. The controller must therefore be modified to exclude states that are not observable.*

Recall the input current developed in the last section

$$\boldsymbol{I}_S = h_S(\boldsymbol{x})^{-1}(a(\boldsymbol{x}) - k^{-1}(\boldsymbol{x})K\boldsymbol{x})$$

where a contains the forces and torques generated by gravity, angular velocity and the permanent magnets. When one of the angular states is unavailable, it renders part of a unknown. More specifically the torques in $p(\boldsymbol{x})$*, which constitutes the three lower entries*

$$p(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{F}_g \\ \boldsymbol{\omega} \times \mathcal{I}\boldsymbol{\omega} \end{bmatrix}.$$

These limitations cause the controller to be unable to compute the full $p(\boldsymbol{x})$ *vector. This leads the feedback linearization to not be quite exact. Usually, this would be a problem, but as shown in the stability analysis below, some measures can be taken to mitigate this. The* $p(\boldsymbol{x})$ *vector the controller can compute is therefore reduced to*

$$p(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{F}_g \\ \boldsymbol{0} \end{bmatrix}.$$

This means that the controller is slightly limited in its ability to control the system, at least to mitigate torques induced by the angular velocity and the moment of inertia.

The other change to the setup proposed earlier is to exclude the non-controllable and non-observable states γ and $\dot{\gamma}$ from the controller. This is intuitively done by excluding the sixth entry in all vectors and matrices involved in the input calculation in (13.12). In addition to this, the corresponding entries in the A and B system matrices are also removed in the calculation of the gain matrix K. Therefore, the total number of states controlled is ten of the system's twelve states.

Stability analysis

This section outlines the tests done to assess the stability of the system with the feedback linearizing controller employed. Analysis was done both theoretically and in practice utilizing simulation of the system in Simulink. This is done to ensure that the assumptions made earlier in the development hold, both theoretically and practically. Firstly, a Lyapunov analysis of the system was done to prove theoretical stability before it was tested in practice through simulation.

Theoretical analysis

Simulink: a block diagram environment for system modeling, included in some MATLAB licenses.

Before any simulated analysis of system stability can commence, it is helpful to assess Maggy's theoretical stability. This way, any inconsistencies in the method deduction can be directly revealed and addressed accordingly, as well as offering some insight into how the system is expected to behave in the simulations. Using Lyapunov analysis, the stability of the system with the developed controller can be proved. For this, a Lyapunov candidate function is chosen as

$$V(\boldsymbol{x}) = \boldsymbol{x}^T R \boldsymbol{x} \tag{13.14}$$

where R is a positive definite and symmetric matrix. Differentiating the Lyapunov function with regard to time gives

$$\dot{V}(\boldsymbol{x}) = \dot{\boldsymbol{x}}^T R \boldsymbol{x} + \boldsymbol{x}^T R \dot{\boldsymbol{x}}. \tag{13.15}$$

Recall the full system description as

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bk(\boldsymbol{x})(h_S(\boldsymbol{x})\boldsymbol{u} + h_M(\boldsymbol{x})\boldsymbol{I}_M - p(\boldsymbol{x})).$$

Inserted into the differentiated Lyapunov function yields

$$\begin{aligned}\dot{V} =& (A\boldsymbol{x} + Bk(\boldsymbol{x})(h_S(\boldsymbol{x})\boldsymbol{u} + h_M(\boldsymbol{x})\boldsymbol{I}_M - p(\boldsymbol{x})))^T R\boldsymbol{x} + \\ & \boldsymbol{x}^T R(A\boldsymbol{x} + Bk(\boldsymbol{x})(h_S(\boldsymbol{x})\boldsymbol{u} + h_M(\boldsymbol{x})\boldsymbol{I}_M - p(\boldsymbol{x}))).\end{aligned}$$

At this point, the developed feedback linearizing controller $\boldsymbol{u}$ can be inserted into the equation. When the full state measurements are available, this

yields a trivially stable case due to the gain matrix K being designed such that $A - BK$ is Hurwitz.

$$\dot{V} = \boldsymbol{x}^T((A^T - K^T B^T)R + R(A - BK))\boldsymbol{x} \tag{13.16}$$

Using the Lyapunov equation

$$A^T P + PA = -Q$$

which holds when P and Q are positive definite and symmetric matrices while A is Hurwitz. This gives

$$\dot{V} = -\boldsymbol{x}^T Q \boldsymbol{x}.$$

This candidate function fulfills the criteria presented in part I for a strict Lyapunov function. This guarantees asymptotic stability for the system. Furthermore, $V(x)$ is radially unbounded, giving a global result. The controller therefore guarantees global asymptotic stability, global in the sense of the entire domain in which the magnets can influence the levitating magnet.

In this system, unobservable states are present in the system description. This means that not all nonlinearities can be eliminated by the controller directly because they are unknown. Therefore, some considerations and assumptions must be made in order to carry out the analysis. Assuming that the unobservable states γ and $\dot{\gamma}$ are internally stable (13.20) allows the assumption of a Hurwitz $A - BK$ to hold. Using the Lyapunov equation on the Hurwitz $A - BK$ in the same manner as before yields the same result, with an added residual term.

$$\dot{V} = -\boldsymbol{x}^T Q \boldsymbol{x} - 2\boldsymbol{x}^T RBk(\boldsymbol{x})p'(\boldsymbol{x}) \tag{13.17}$$

where $p'(\boldsymbol{x})$ is the part of $p(\boldsymbol{x})$ the controller cannot cancel

$$p'(\boldsymbol{x}) = \begin{bmatrix} \mathbf{0}_{3\times 1} \\ \boldsymbol{\omega} \times \mathcal{I}\boldsymbol{\omega} \end{bmatrix},$$

$$\dot{V} = -\boldsymbol{x}^T Q \boldsymbol{x} - 2\boldsymbol{x}^T \underbrace{RB\gamma(\boldsymbol{x})}_{0<} p'(\boldsymbol{x}). \tag{13.18}$$

This boils the problem down to analyzing this portion of the residual part

$$\begin{bmatrix} \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix} \begin{bmatrix} \underbrace{\boldsymbol{\omega} \times \mathcal{I}\boldsymbol{\omega}}_{p'(\boldsymbol{x})} \end{bmatrix}, \tag{13.19}$$

with $\mathcal{I}$ being $\begin{bmatrix} \mathcal{I} & 0 & 0 \\ 0 & \mathcal{I} & 0 \\ 0 & 0 & \mathcal{I}_\gamma \end{bmatrix}$ *owing to the symmetry of the levitating magnet.*

$$\begin{bmatrix} \dot{\alpha} & \dot{\beta} & \dot{\gamma} \end{bmatrix} \begin{bmatrix} \dot{\beta}\dot{\gamma}(\mathcal{I}_\gamma - \mathcal{I}) \\ -\dot{\alpha}\dot{\gamma}(\mathcal{I}_\gamma - \mathcal{I}) \\ 0 \end{bmatrix} = 0 \tag{13.20}$$

This shows that as long as α and β together with their derivatives are controlled, γ will not destabilize the system, at least as long as a torque is not specifically applied to this axis of rotation.

This means that any perturbations in the state measurements will result in equal and opposite perturbations within the residual term, essentially giving

$$2\boldsymbol{x}^T RBk(\boldsymbol{x})p'(\boldsymbol{x}) = 0 \tag{13.21}$$

and thus canceling the residual part, leaving only

$$\dot{V} = -\boldsymbol{x}^T Q\boldsymbol{x}. \tag{13.22}$$

From this, the same conclusion as for the full-state feedback case can be drawn.

However, this analysis does not consider potential limitations on the controller. Constraints on the input might influence the controller's ability to impose convergence on the state, and hence the system's stability. A concrete example of this is the conclusion above. While a controller with no input limitations will have no problem with the residual term, a controller with limited input might not be able to control the angular states to a large enough degree. This can lead to instability in the system, not necessarily because of the residual term directly, but because the angular states will not converge quickly enough, making controlling the system near impossible.

Pole placement

The poles of a system determine its stability and response properties and can be used to manipulate a system's dynamics through pole placement. The feedback linearizing controller employs a linear pole placement control scheme, the poles must therefore be chosen manually. There are many ways to choose these poles, and the choice depends on what properties the controller should have. A stable system's poles are placed in the left-half plane, which means that their real parts are negative. The poles can also have complex parts, which gives the system a more powerful response. This often causes the system to experience overshoot and oscillations when controlling the system.[13]

[13] Trond Andresen, Jens G. Balchen, and Bjarne A. Foss. *Reguleringsteknikk*. Institutt for teknisk kybernetikk, NTNU, 2016. ISBN 9788278422021

The vector of poles consists of ten entries, one for each state

$$poles = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} \end{bmatrix}.$$

Before any stability testing in Simulink can commence, these poles must be chosen to promote stable behavior of the system and the states to be controlled. The main focus of the control scheme is to control the system in the x, y and z directions, that is radial placement and height of the levitating magnet. Therefore one might think that the poles corresponding to these states are the only ones that matter, but the angular states are just as important.

Since the controller has been modified to omit the torque generated by the angular states, any big deviations in the angular states will destabilize the system when constraints are put on the input. Therefore it is essential to maintain control over these states to be able to control the translational states. In the case of this system, a set of different poles was found to stabilize the system, with different constraints placed on the input. A set of ground rules can be laid down for tuning the poles

1. *Establish some sort of initial stability, that is find poles that semi-stabilize the system. This should not be too hard as long as the poles are in the left-half plane.*
2. *Tune the poles for the angular states first in order to achieve convergence in these states. This entails tuning p_4, p_5, p_9, p_{10}.*
3. *When the angular states reach convergence, tuning of the translational poles becomes much easier since deviations in the angular states do not destabilize the system.*

The method above identified three sets of poles for the system through trial and error. One for the system without input constraints and two for the system with different constraints. The first two constraints are quite loose, while the last two are much tighter. For the physical implementation of the system, these constraints must be tightened further, which will involve more modification of the poles.

When dealing with limitations such as imposed constraints, it is first easier to tune the poles of the unconstrained system. This is due to the fact that it is able to deal with the deviations in the $p(\boldsymbol{x})$ just fine. These poles can then be used as the initial stable poles described above, from which new sets of poles for the constrained system can be found.

Table 13.1 shows the developed pole configurations referenced above. Generally, the controller is tuned by changing the last five poles to a larger degree than the first five. This is because the first five states are directly impacted by the five last, meaning that the first states are more or less controlled by the last. Some importance must of course be placed on the first five

Constraint	$[p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}]$	**Note**
–	[-0.5,-0.5,-10,-4,-5,-2000,-2000,-9000,-9,-1]	Convergence in X,Y,Z
±5000 and ±1000	[-0.5,-0.5,-10,-4,-5,-2000,-2000,-9000,-9,-1]	Convergence in X,Y,Z. Some deviation in Z.
±150 and ±50	[-0.5,-0.5,-2.5,-2.5,-10,-2000,-2000,-9000, -15,-12]	Convergence in X,Y,Z. Some deviation in Z.

Table 13.1: Different pole configurations for different input constraints.

states since they are the ones we actually want to control, but generally, the last five states were found to be the most influential.

The values of the poles in this table reflect the theory discussed earlier, as can be seen in the increase in p_9 and $p,_{10}$ respectively. By increasing the magnitudes of these, the controller is encouraged to promote convergence in the angular states more quickly, preventing any deviation between the controller's model and the real system. Decreasing the pole value for z might seem counterintuitive, but it allocates more of the input to deal with the angular states at the cost of some deviation in the z-direction. This trade-off is not optimal, but it did show to produce satisfactory stability anyway.

As outlined through Table 13.1, adaptation of the poles is paramount when imposing restrictions on the input. To further tighten the constraints, the poles must be modified again, but doing this in a way that does not compromise stability has proved to be complicated. Furthermore, the poles must not be modified in a way that further introduces more deviations into the system, as that can also promote instability.

Method for simulated stability analysis

Using simulation to test the system stability is a very powerful tool, but in order to utilize it as efficiently as possible a method for running the tests might be useful. Having deduced a set of poles for the system in different configurations allows for stability analysis directly through a simulator. This is done to evaluate the results of the Lyapunov analysis. Through simulation of the system, a test can be developed to approximate a region of convergence for the controlled system, in which the performance of the controller is inherently visible. This approach also allows for the impact of various disturbances and input limitations to be closely inspected.

Before presenting the method in detail, we propose a simulator setup in Simulink to run all the tests on. The setup is based on the simulator

developed for the LQR controller, with changes made in accordance with the system changes presented earlier in Chapter 13. Figure 13.9 shows

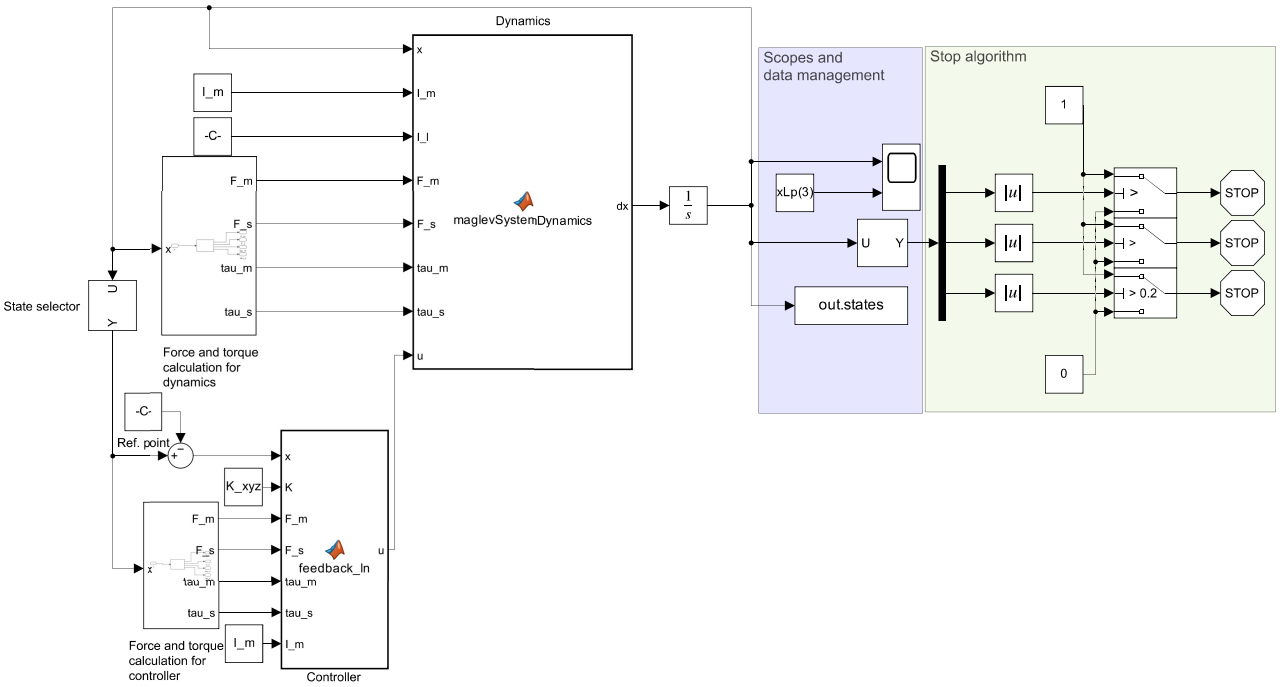


Figure 13.9: Simulink setup

the setup as it is in Simulink. There are two main blocks responsible for the system dynamics and the controller calculations, respectively. Two subsystems calculate the force and torque for both the system dynamics and the controller, given that the controller does not have access to all the states. An algorithm for the simulation to stop when the states go beyond some predetermined boundaries is also implemented.

The tests were conducted by drawing lines radiating from the origin of a plot showcasing radial deviation on the x-axis and deviation in height on the y-axis. Code was developed that would test the stability of the system in the endpoints of these lines, if the performance of the system in these points did not satisfy the stability criterion the next point to be tested was halfway between the origin and the current point. When a stable point was found along the lines, the code would move on to the next line and repeat the cycle until all lines had been tested.

Figure 13.10 showcases an illustration of the test procedure, with points to be tested indicated with red circles. For each point tested along the lines, multiple simulations are run to ensure that the convergence is tested in as many directions of radial displacement as possible. Specifically, a pre-

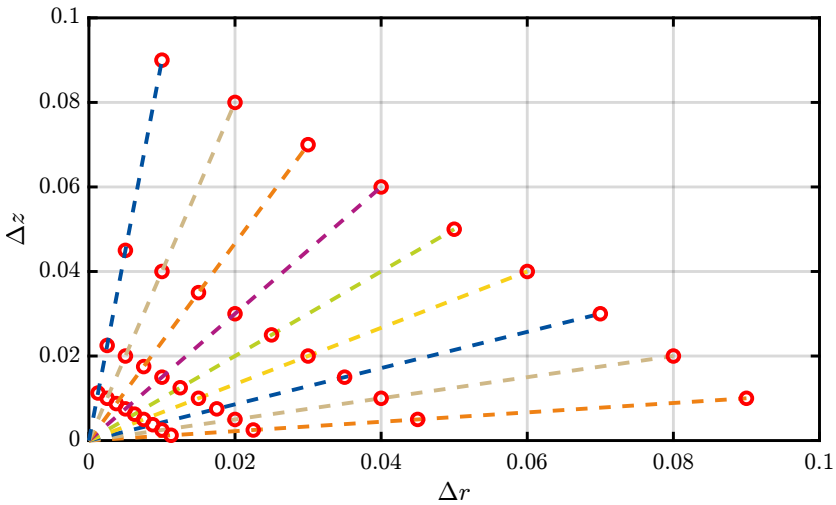


Figure 13.10: Illustration of the test procedure for approximating regions of convergence

determined number of points are chosen along the circumference of a circle with Δr radius and tested individually. In order for a point to be designated as stable (or convergent), all points must produce a stable outcome.

This procedure generates a hull of the region of convergence, from which additional tests can be carried out to further ensure its legitimacy. In most cases, these tests were conducted on new points along the boundary of the hull, but they can also serve to bolster the approximation by testing multiple points within the hull as well.

For the test to be as efficient as possible, a meaningful set of points must be tested such that the result is productive and informative. In order to determine which points are productive and which are not, looking at the system at hand is important, more specifically, the forces involved in the magnetic levitation. Central to this endeavor is providing some sort of measurement of the magnetic forces emanating from both the solenoids and the permanent magnets.

*As described earlier (**???????**), we can say something about the magnetic field in a point. This knowledge about the magnetic field vector $\boldsymbol{B}$ can be used to say something about its strength in that same point. The magnitude of the magnetic field vector, $|\boldsymbol{B}|$, provides the strength of the current field.*

Figure 13.11 shows the heatmap of the magnetic fields in a layer just above the base. In this figure, only the fields from the permanent magnets are visible, since the input is set to zero. These heatmaps can also serve as a validation of the model parameters used in the simulator by comparing it to the system illustration in figure 5.1.

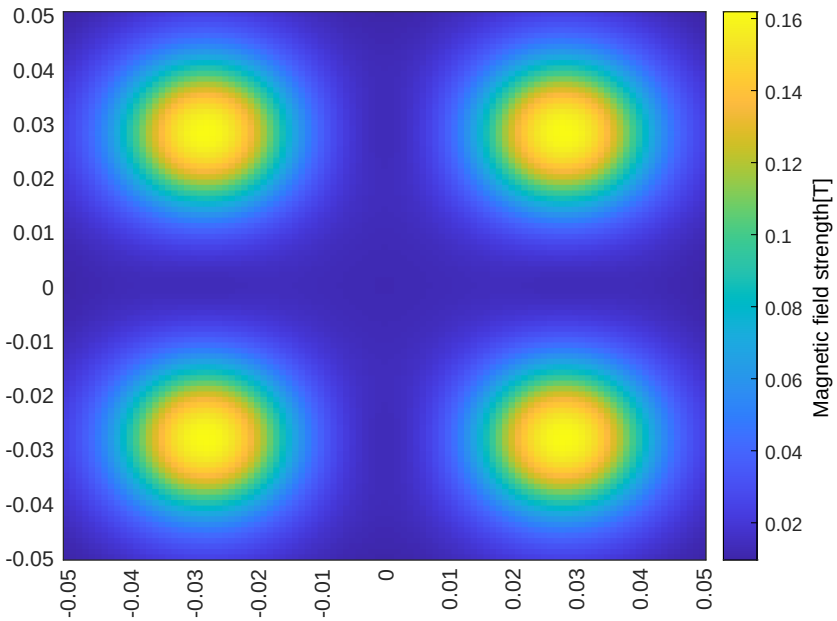


Figure 13.11: Heatmap of the magnetic forces at $z = 0.015$ and $u = 0$

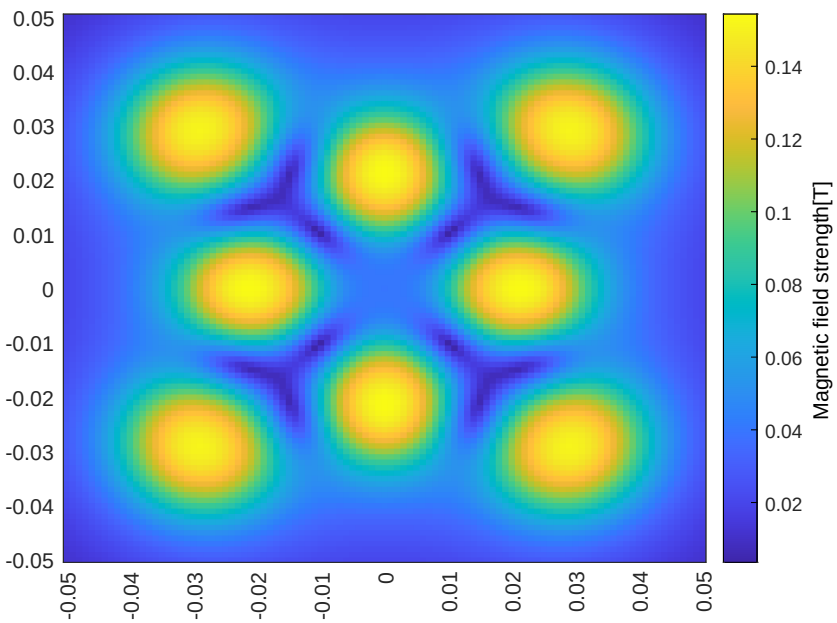


Figure 13.12: Heatmap of the magnetic forces at $z = 0.015$ and $u = 15$

Figure 13.12 shows the same heatmap as above when an input is applied to the solenoids. These two heatmaps serve to highlight a clear symmetry in the system with regard to the distribution of the magnetic fields. This can be used to deduce good points to test in the stability analysis. The symmetric nature also allows for more detailed tests without necessarily increasing the computational complexity, since it makes testing points along the entire

circumference redundant.

Focusing testing on points in only one of the quadrants allows for more precise and efficient approximations by partitioning the problem and assuming that the results in one of the partitions will hold in all of them. This increases the total resolution of the test without the simulation time also following suit. Choosing the upper right quadrant as the main focus for testing yields the specific points of testing showcased in Figure 13.13, with points both on and between axes.

As evident from this figure, the total resolution of the tested points is much higher than the actual points tested. In this case, the total points accounted for are 16, while only 5 of them are actually tested. The resolution of the tests can be further enhanced by increasing the number of points tested in the quadrant. In addition to this, the quadrant can be further divided in half from 90° *to sectors of* 45° *to provide an even higher resolution. In this case, another assumption must be made; that stability in one of the sectors is equivalent to stability in its counterpart, in which the magnetic fields are mirrored.*

Stability analysis through simulation

A good measure of the system's stability is the region of convergence. This is the region in which the system is expected to return to the set point after a disturbance or deviating from the initial conditions. Using the method outlined above, approximations of these regions can be made. These approximations are not necessarily ground truths, but relative to each other they tell an important story about the system's overall stability.

Using the approximation of the region of convergence for the LQR as a reference provides a good base for presenting the feedback linearizing controller's properties. It also gives an initial impression of the system's overall stability and is a good starting point when discussing the differences between the two controllers. These tests are done with poles placed in accordance to Table 13.1, and the set-point assigned to the equilibrium along the z-axis in the center of the base. The grey dotted line shows the placement of the solenoids along the radial axes.

$$x_{lp} = [0, 0, 0.0365, 0, 0, 0, 0, 0, 0, 0, 0]$$

The logic behind the choice of set-point is the same as for the earlier presented controllers, to provide a good basis for stability for the controller.

Figure 13.14 shows the approximations of the regions of convergence for the LQR controller and the feedback linearizing controller, respectively. At

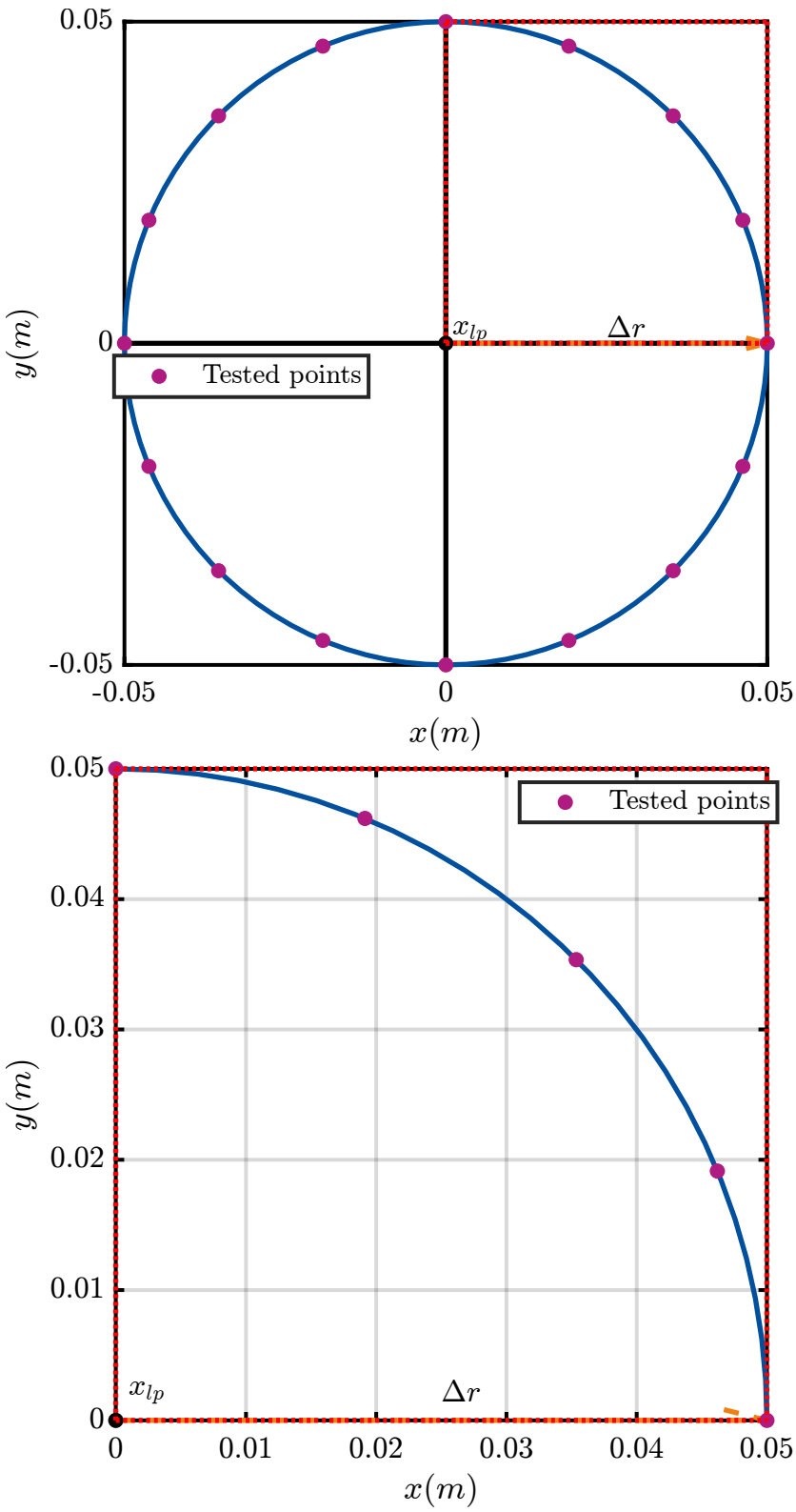


Figure 13.13: Illustration of points tested along the radial displacement

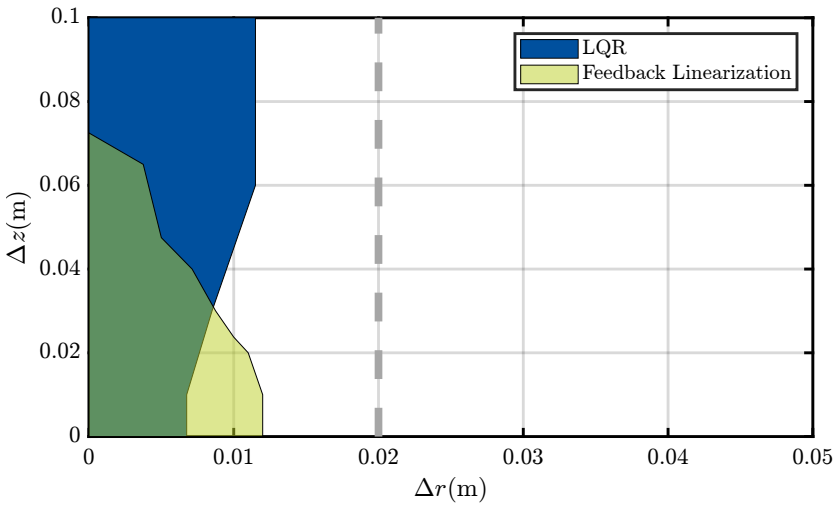


Figure 13.14: Comparison of Regions of Convergence between LQR and Feedback Linearization

first glance, the LQR controller seems like the most stable of the controllers, offering a larger region of convergence. The most notable difference between the two is the LQR controller's superior ability to handle deviations along the z-axis while also being able to handle some deviation in the radial direction. The fact that the LQR controller handles the deviations in height so well might indicate that the system behaves more linearly along the z-axis than along the x- and y-axes. This makes sense when taking the symmetry of the system when the levitating magnet is centered into account. Controlling the magnet in this scenario can be more or less equated to balancing the forces along the z-axis, which is a linear problem.

An interesting aspect of the LQR controller's region of convergence is the fact that it is able to handle radial deviations better the higher above the base the magnet is. There can be many reasons why this is the case. For example, it might be easier for the controller to control the magnet towards the center of the base whilst it is "falling". Thus being able to utilize its velocity together with minor torques applied to the magnet to steer it centrally rather than having to generate both torque and velocity by itself.

A central part of the controller implementation on Maggy is the topic of input limitations. To mimic the real system more closely, constraints have been put on the controller in the simulation. Obviously, this will impact stability, reducing the region of convergence, but the question is by how much?

Figure 13.15 shows the impact of imposing constraints on the input of the system. As expected, these limitations reduce the region of convergence. What is most interesting about these plots is how the reduction in the re-

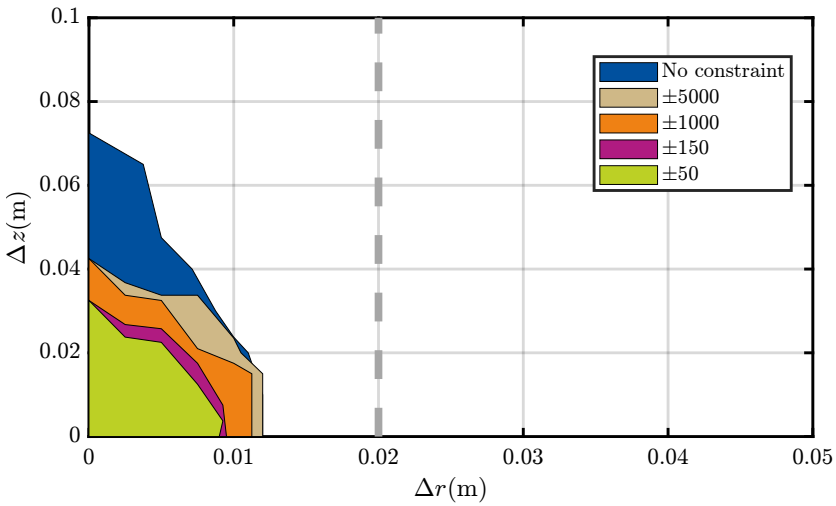


Figure 13.15: Comparison of Regions of Convergence for different input constraints

gions along the z-axis is generally larger than in the radial directions. This comes as a natural consequence of the limitations, where the solenoids are not able to produce enough "power" to control the magnet at higher positions. It is promising that much of the controllability in the radial directions seems to be retained. This means that while the solenoids are weaker, the controller is still able to cancel much of the nonlinearities. While the constraints are still too high with regard to a physical implementation, having been able to tighten them from unlimited down to 50 bodes well for further development.

Another important aspect is how the system reacts to any delays in data transfer. By placing a transport delay-block in the Simulink block diagram, the delay of data delivery between the controller and the physical system can be simulated. The simulation itself was done using the controller without any constraints.

Figure 13.16 details the influence of delays on the controller's region of convergence. The delays do impact the region of convergence quite visibly, even with just small delays. The actual delay in the physical system will consist of both transport delay and the delay caused by the controller's computation time. Depending on which manner of implementation is employed, the delay is expected to be higher than 0.15ms, therefore a further reduction in the region of convergence is to be expected. Most interesting in this case is the fact that much of the controllability along the z-axis is retained, while the loss of controllability occurs mostly in the radial directions. This is a natural consequence of the delay, diminishing the controller's ability to deal with the nonlinearities. As discussed earlier in this section, there are more nonlinearities associated with radial translation than movement along the

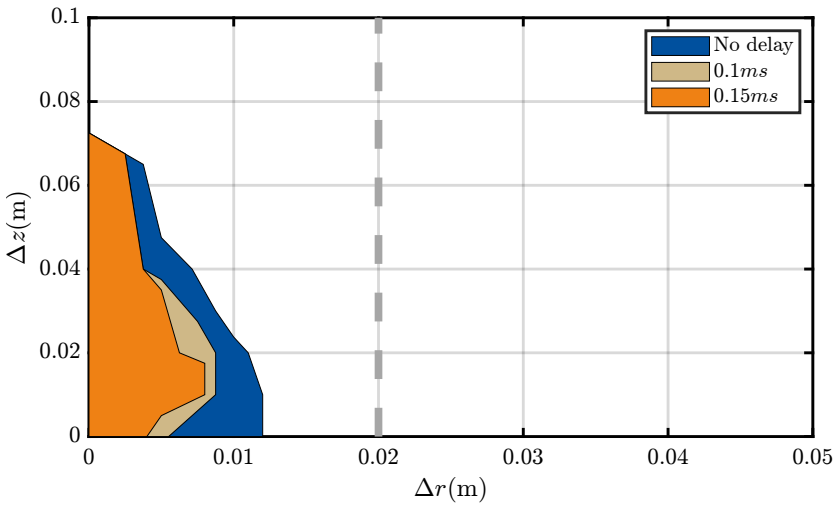


Figure 13.16: Comparison of Regions of Convergence for different transport delays

z-axis.

Remarks on stability

When inspecting the approximations of the regions of convergence, what is immediately apparent is how surprisingly good the LQR controller is. From this, the natural conclusion is to use the LQR controller to control the system. One aspect of the LQR controller not yet discussed is how it compares to the feedback linearization when limits are imposed upon the system. This is the catch with the LQR controller, it cannot handle any delay in data transfer between the system and the controller. This makes very much sense when considering its linear nature, any delay brings the system further away from the operating point before the controller can steer it back.

This might also be related to the system's configuration. During the development of the Maggy maglev system, there have been multiple iterations in which the configuration of magnets and solenoids has been different. The stability tests carried out in this analysis were done using the current configuration, namely Maggy V2.6, with parameters included in the parameters_maggy_V2.m-file in the GitHub repository[14]*. When tested on one of the earliest configurations where the solenoids and the permanent magnet were switched relative to the current configuration, the LQR was at least able to handle some delay.*

[14] https://github.com/Hansolini/Take-home-Maglev-lab

This implies that there are different stability properties affiliated with different magnet configurations. Figures 13.17 and 13.18 illustrate the oldest and newest configurations of the system, respectively. It is clear that these

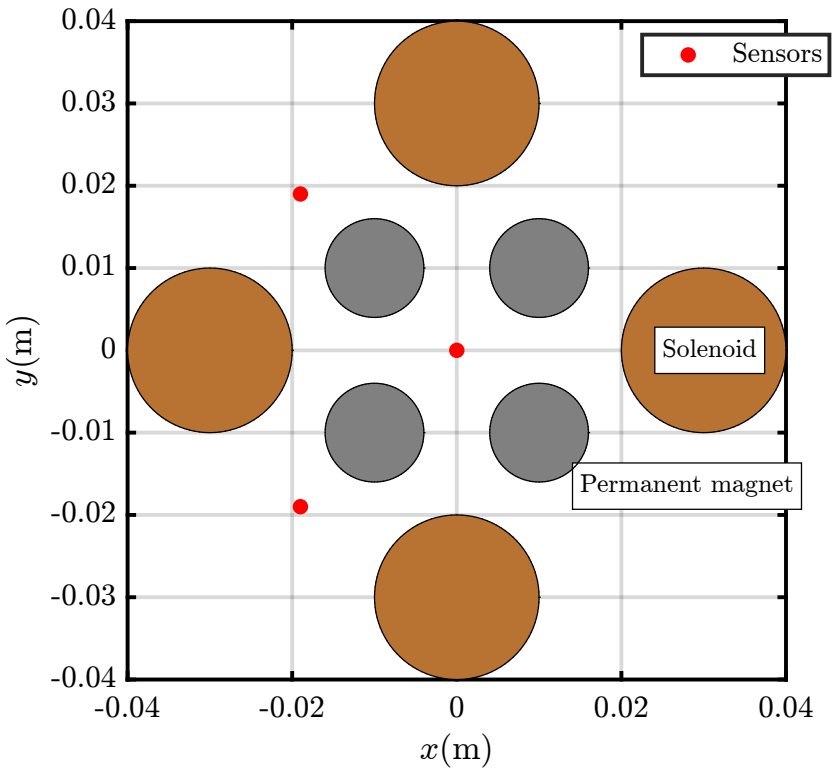


Figure 13.17: Illustration of the earlier configuration

two configurations provide different conditions for stability, given that they are more or less each other's "inverses". The first configuration will favor systems where the solenoids are stronger than the permanent magnets because they will have to produce stronger magnetic fields to influence the levitating magnet. The current configuration will favor stronger permanent magnets while putting the responsibility of stabilization wholly on the solenoids. This happens due to their central placement, giving them better conditions to adjust the magnetic fields closest to the levitating magnet. The smaller size of the permanent magnets in the first configuration also means a less stable base. The area over which the levitating magnet can be controlled is also smaller.

Another detail to be addressed with the regions of convergence is the superiority of the LQR controller over the feedback linearizing controller in controlling the magnet in the z-direction. It seemed like the LQR controller could deal with very large deviations along the z-axis, too large to actually be realistic and therefore not included in the approximations presented earlier. One can even argue that 10cm, as included in the plots, is too much of a deviation to actually be realistic.

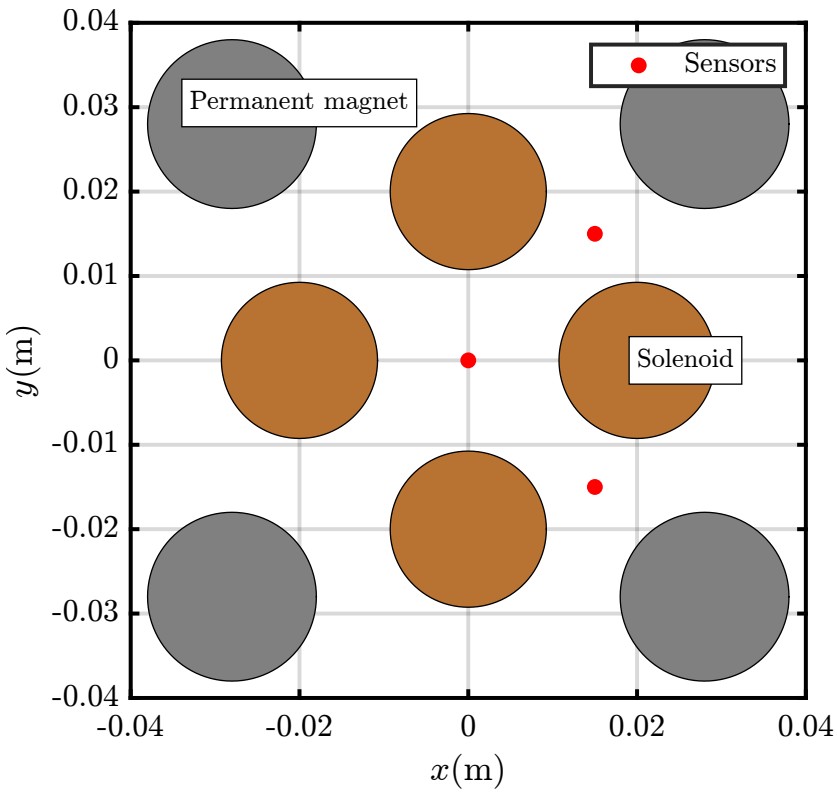


Figure 13.18: Illustration of the magnet configuration of Maggy V2.6

Chapter 14

Controller implementation

Now that the controller has been presented and tested sufficiently, it is time to investigate different modes of implementation. How the implementation is handled is very important, as it provides the very base on which the controller will operate. A bad implementation will consequently have implications on many aspects of the system, first and foremost its stability. There are multiple ways of doing the implementation, but they can be boiled down to two specific methods; on-chip directly on the Teensy and off-chip by using serial communication for data transfer.

Of course, the easiest and most intuitive way is to try to do all computations directly on the Teensy itself. This on-chip approach assumes that all code can be translated and adapted to work in the Arduino environment. This method offers the minimum amount of delays for data transport from the sensors to the actuators. The problem with this approach is the assumption that the system forces and torques can be modeled directly on the Teensy, which might not be feasible. Another worry is that the modeling will be too time-consuming and that any time saved in transport delays will be lost to computation alone.

The other approach is doing all computations off-chip using serial communication between the Teensy and some other device. This allows for more flexibility when it comes to choosing how to solve the modeling and computation. A wide range of software and programming languages are available for computers that can solve the equations related to the controller much more efficiently than any self-coded Arduino sketch. Of course, the potential problem with this method is the opposite of the on-chip approach: the delay introduced by data transfer will become too big, causing the sampling rate to become too low for stable levitation.

Implementation study: off-chip computations

The first alternative we consider is off-chip computations. Since a simulator has already been developed for the system using MATLAB, MATLAB is the software of choice for this endeavor. The structure of this implementation will closely resemble the structure of the simulator to match the conditions of the simulated stability analysis as much as possible.

Structure

It then makes sense to introduce the intended structure of the system with the controller employed. This structure is best visualized through a flow chart, where all the different functions and mechanisms can be easily modeled as separate boxes.

Figure 14.1 shows a rough outline of how the MATLAB implementation is structured. Each block represents a function whose functionality is described within. This implementation offers a pretty streamlined structure, and serves as a good starting point for the development of the Arduino implementation as well. To underline its resemblance to the simulator, the Plant box can be replaced with the MATLAB functions for dynamics and measurements to obtain the simulator's structure.

Communication speed

Communication between the Teensy and MATLAB will be handled through serial communication, of which the basics are briefly covered in Part III. The main focus is on measuring any time delay caused by the transmission of data between the two. Serial communication in MATLAB is handled through the serialport-object, with separate write and read functions.[1,2] *The main obstacle is that we expect the Teensy to be far superior to MATLAB with regard to computational time, given that Teensy runs on pre-compiled code and is specialized for speed.*

[1] Inc. The MathWorks. serialport, f. URL https://se.mathworks.com/help/matlab/ref/serialport.html

[2] Inc. The MathWorks. Write and read serial port data, e. URL https://se.mathworks.com/help/matlab/import_export/write-and-read-serial-port-data.html

While the MATLAB documentation does not mention any properties related to transfer speed or any delays, it makes sense to test this. Recall the data transmission speed test outlined in Part III, using Paul Stoffregen's custom code.[3] *The same method can be utilized to test the transmission speed between two devices or more specifically, two applications. By examining the loop time of the Teensy, as well as the total time it takes for a given number of loops, a measure of the performance of MATLAB relative to the Teensy can be established.*

[3] Paul Stoffregen. Usb-serial-print-speed-test. https://github.com/PaulStoffregen/USB-Serial-Print-Speed-Test/tree/master, 2019

The method can be summed up in four steps:

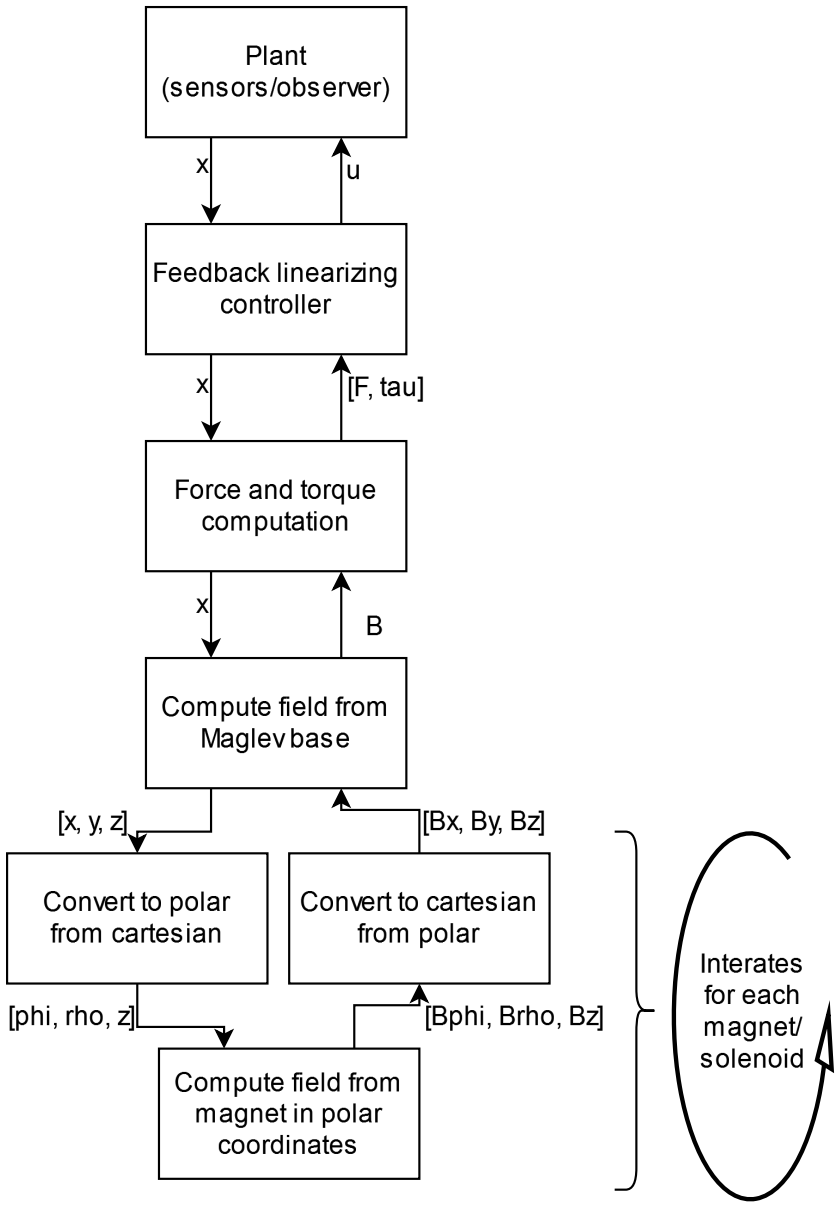


Figure 14.1: Flowchart of the structure of the MATLAB implementation

1. *Establish the time the Teensy uses to print 500000 lines from the code.*
2. *Establish the time MATLAB uses to receive 500000 lines, without printing them to avoid any delays from console operations.*
3. *Find the relation between the measured times by* $\frac{T_{MATLAB}}{T_{Teensy}}$.
4. *Use this relation and the Teensy loop time to find a suitable sample time to synchronize the two. This will reflect the true data transmission speed.*

```
teensyRec = serialport("COM6",1000000);
```

```
i=0;
while true
    r = read(teensyRec,32,"string");
    if i==0
        tic;
    end
    if i==500000
        toc
        r
        break
    end
    i = i+1;
end
delete(teensyRec)
```

The code snippet above opens the serial port COM6, which is the one the Teensy is configured to communicate through at a baud rate of 1000000. Recall from Part III that the baud rate does not impact the transmission speed of the Teensy, but the Matlab function serialport needs it to function properly. Data of type string and length 33 bytes are then read from the opened port a given number of times before the serial object is deleted.

By making some adjustments to the Teensy code from Paul Stoffregen's repository, we can measure the time the Teensy uses to loop through the code a given number of times.[4]

[4] Paul Stoffregen. Usb-serial-print-speed-test. https://github.com/PaulStoffregen/USB-Serial-Print-Speed-Test/tree/master, 2019

```
void setup(){
    start = micros();    //Start time
}
void loop(){
    if(count == 10500000){     //Stop after 500000 iterations
    start = micros()-start;    //Time elapsed
    printtime=micros();
        while(true){
            if(micros()-printtime>5000000){   //Print every 5s
              Serial.println(start);
              printtime=micros();
            }
        }
    }
}
```

Initially, the test measures the time for 500000 iterations, which was measured to around 1 second. Now the next step is to find out how "out of sync" the transmission to MATLAB is. This is done by using the code presented earlier in this section and timing the time it takes for MATLAB to receive 500000 lines. This time is measured to be around 53 seconds, which means that MATLAB is very out of sync with Teensy.

The relation between the two running times is calculated as $\frac{53s}{1s} = 53$*, meaning that the Teensy transmits data around 53 times faster than MATLAB is able to receive it. To then be able to measure the true transmission time, the Teensy code must be sampled by some frequency such that it only transmits when MATLAB can receive. To determine this frequency, or sampling time, the running time for one iteration of the Teensy code must be determined. This can also be done by amending the code in the same manner as earlier.*

```
void loop(){
    unsigned long iterationtime = micros();
    if(count == 10000002){  //Stop after 1 iteration
        printtime = micros();
        while(true){
            if(micros()-printtime>5000000){
              Serial.println(iterationtime-start);
```

```
            printtime = micros();
        }
    }
}
```

With this setup, the iteration time for one loop was measured at $\sim 2\mu s$*. This implies that if we impose a sample interval of* $53 * 2\mu s = 106\mu s$*, the transmitter and receiver should be synchronized. With this sample interval the Teensy uses 54 seconds on 500000 iterations, while MATLAB also uses 54 seconds to receive 500000 lines. From this, we can now say something about the total transmission bandwidth.*

Using a sample interval of $106\mu s$*, this is the output from MATLAB after receiving around 500000 lines*

```
r = "count=10500869, lines/sec=9249"
```

Which gives

$$\begin{aligned} 9249 lines/sec * 31 bytes/line &= 286719 bytes/sec \\ 286719 bytes/sec * 8 Bits/byte &= 2293752 Bits/sec \\ &\approx 2.3 MBits/sec \end{aligned}$$

While this is a drastic decrease from the native Teensy measurements, which gave 143,7 Mbits/sec, this should be able to handle a sample frequency of just under 5kHz, since data must go both ways. The code for this test is available under student_contributions/code /PIDKF_master/Implementation_off_chip/Transmission_Speed_Test in the GitHub repository[5]*. The repository is also included as an attachment to this thesis.*

[5] https://github.com/Hansolini/Take-home-Maglev-lab

Since data must be sent back and forth between the two applications, some logic must normally be in place to decide when to transmit and when to listen. This is where the dual serial capability of the Teensy comes in. By opening two virtual serial ports, each with its own name, we can assign one channel for transmission to the Teensy and another for transmission from the Teensy. This only entails opening two COM ports in MATLAB, as opposed to only one.

Communication structure

The next obstacle to tackle is data formatting. How should the data be structured, both with regard to transmission speed, but also robustness? Table 14.1 shows the length of the bit representations for four different datatypes that might be interesting in this case. Note that a string is treated as an array of chars, which in turn is stored following the ASCII encoding standard.[6]

[6] Arduino. char, 2019. URL https://www.arduino.cc/reference/en/language/variables/data-types/char/

Data type	Memory size
int	4 bytes
float	4 bytes
double	8 bytes
char	1 byte

Table 14.1: Memory size for some useful data types

For transmission of measurement data from the Teensy, the data must be as accurate as possible, at least down to the millimeter level. Float offers a precision of about seven decimal digits, which suits the controller's needs very well. As discussed earlier, we anticipate that ten states will be estimated, so ten different data entries must be transmitted. This results in a total data length of 40 bytes. With a transmission speed of 2.3 Mbit/sec, this means a total transmission time of $\approx 140\mu s$.

For transmission of the inputs to the Teensy, decimal points are not needed as they are formatted as PWM-signals in the interval ± 255. *The natural choice here might seem like int, and it may well be, but in a bid to try to reduce the transmission time, it was decided to format them as strings. Since we do not always know the exact length of each entry, this may reduce the total length of the transmitted data in some circumstances.*

For example, in the worst case, the total bit length will be 17 bytes, including delimiters. In the case where every entry only consists of two digits, the total length will be 11 bytes. This might seem like a pointless move, but in the long run, where the levitating magnet is in a stable mode, the inputs will hardly be at max capacity, meaning less data to be transmitted.

Computation time

As a last endeavor in this section, the computation time of the controller itself must be investigated. This will be added to the delay produced by the data transmission to estimate the total delay and, from that, deduce a total sampling time for the entire controller.

By implying a suitable sample time, the test will also assess the implementation's suitability regarding stability. This is done through the placement of timers at certain points in the MATLAB code using tic/toc.[7,8] *In the end, the main goal is also to compare the total computation time and transport delay for both implementations. To generalize the test somewhat, the results are the mean of 1000 different runs, each time with randomized state values within some predetermined boundaries, detailed in Table 14.2.*

[7] Inc. The MathWorks. tic, h. URL https://se.mathworks.com/help/matlab/ref/tic.html

[8] Inc. The MathWorks. toc, i. URL https://se.mathworks.com/help/matlab/ref/toc.html

The timers are placed such that the force and torque calculations and the input calculation are timed independently, and by adding them, the total

State	Boundary
x	± 0.05
y	± 0.05
z	± 0.05
α	± 1
β	± 1
$\dot{x}$	± 0.015
$\dot{y}$	± 0.015
$\dot{z}$	± 0.015
$\dot{\alpha}$	± 0.25
$\dot{\beta}$	± 0.25

Table 14.2: Table detailing the boundaries set for the random generation of state estimations

computation time can be obtained. The colored chart in Figure 14.2 details the specific timed portions of the MATLAB controller.

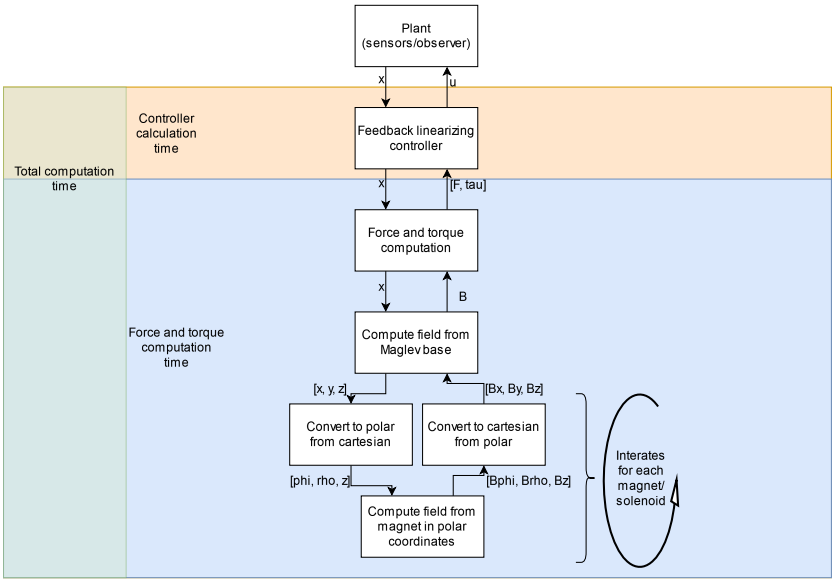


Figure 14.2: Illustration of the different timed portions of the MATLAB controller

This code is run in parallel with a Teensy code which generates ten random data entries within the boundaries in Table 14.2, as well as receiving the string of inputs sent from MATLAB. This enables the test to somewhat simulate an actual running of the system. The code for this test is available under student_contributions/code/PIDKF_master/ Implementation_off_chip/Data_sending_test_final in the GitHub repository[9]. The repository is also included as an attachment to this thesis.

[9] https://github.com/Hansolini/Take-home-Maglev-lab

Figure 14.3 details the total computational time and transport delays associated with the controller using the off-chip implementation. The receive

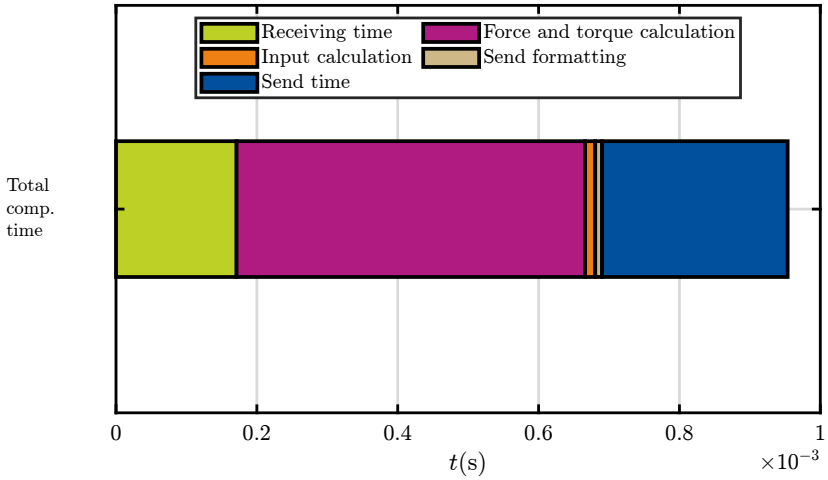


Figure 14.3: Bar graph of the MATLAB running time with separate components. This is from MATLAB's "point of view"

and send times presented here are measured from MATLAB's perspective. These results are somewhat in line with what can be expected with regard to the transmission times. The send time is a little larger than what might be expected, but still within reasonable range of the time calculated earlier. The send time is much larger than what could be expected, which might be a result of the way MATLAB handles outgoing data.

In order to be as efficient as possible in the Teensy code when sending the measurements, the method Serial.write() is used instead of the traditional Serial.print(). Serial.write() differs from Serial.print() in that instead of writing readable characters (ASCII) to the serial port, it writes the binary representation directly.[10]

[10] Arduino. Serial.write(), 2023. URL `https://www.arduino.cc/reference/en/language/functions/communication/serial/write/`

This basically means that instead of doing pointless conversions between numbers and their bit representations, they are sent directly as bit sequences. To achieve this, pointers are used to obtain the bit arrays of a given number directly from its 'address' in the memory, meaning no conversion is needed. This is not the case in the MATLAB implementation, where the serial port object itself casts the data passed to it into the desired data type.

Pointer: an object in computer science that stores the memory address of a variable.

As shown by this graph, most of the time is made up of the force and torque calculations. Since this code is already pretty optimized it means that most of the time reduction must come from an optimization of the transfer times. With this in mind, it might be useful to consider the scenario where all computations are done directly on the Teensy, eliminating transmission delays. And who knows, the pre-compiled Arduino code might run faster than MATLAB after all.

Example: off-chip computations

Recall the PD controller presented in Chapter 13. This controller was originally implemented solely on the Teensy, with all computations being done locally. To assess the limitations of the off-chip approach using MATLAB, this already working controller can be adapted to send its raw data to MATLAB and receive the computed input over serial communication.

To do this, all data calculation is moved from the Teensy to a specialized MATLAB script that can receive and send data. The only thing left on the Teensy is the code strictly meant for data collection through the sensors, the code for receiving the inputs, and the code for input allocation. In theory, this setup should allow for real-time data transfer between the Teensy and MATLAB while the magnet levitates.

Regrettably, the sampling frequency did not reach the heights that allow for stable levitation. After a levitation period of 10 seconds, 27156 data points were generated. This results in a data transmission time of $\frac{10s}{27156} \approx 370\mu s/datapoint$*, and a sampling frequency of 2,7kHz. Which is entirely in line with the findings presented in figure 14.3.*

Despite not being able to stabilize the levitating magnet, this endeavor allows for the robustness of the data transfer to be inspected closer.

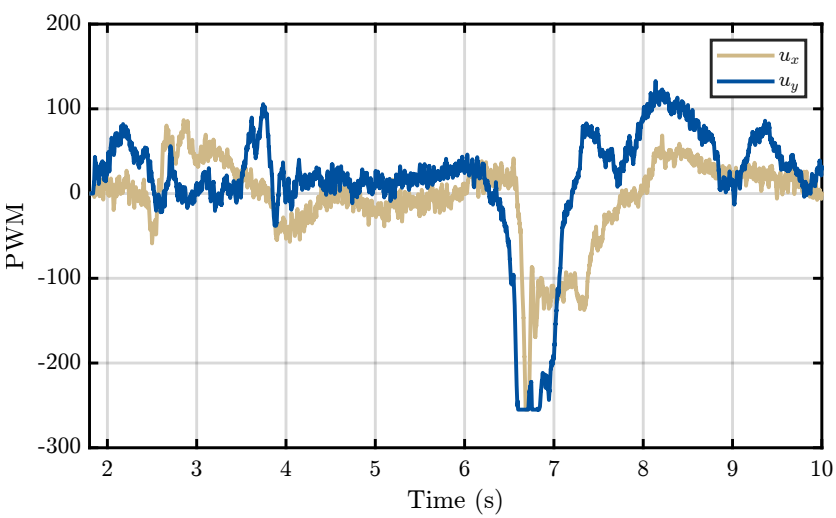


Figure 14.4: Input signal u calculated by MATLAB

Figures 14.4 and 14.5 show the input u_x *and* u_y *as sent from MATLAB and received by the Teensy over a period of 10 seconds, respectively. It is safe to say that the integrity of the data is pretty much retained throughout the transfer period, with the exception of a few discrepancies.*

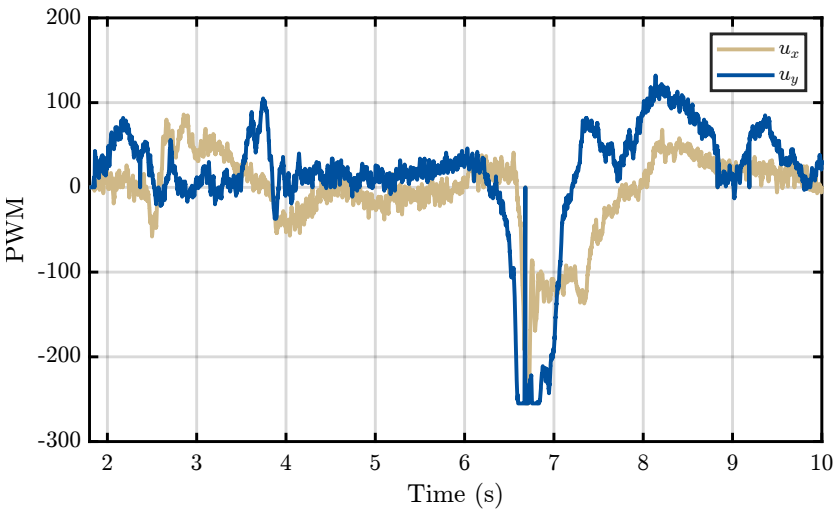


Figure 14.5: Input signal u as received by the Teensy

This serves to prove that the off-chip implementation has a good chance of working, at least from a data integrity point of view.

Implementation study: on-chip computations

With the analysis of the off-chip approach done, the next step is to assess the feasibility of an implementation directly on the system. This requires the controller to adapt to the variant of C++ that the Arduino IDE uses. The controller needs complete calculations of the forces and torques involved, which means that all the relevant functions from the MATLAB implementation must be adapted to work on the Teensy.

We follow the same logic as the other implementations when it comes to structure. Try to make it as similar to the simulator as possible, to provide the same conditions as the stability analysis had.

Structure

The intended structure is similar to that of the simulator, and thus also the off-chip implementation, which allows us to utilize the flowchart shown in Figure 14.1 as a reference. Starting at the bottom of this flowchart allows the development to follow the structure closely, with each function block in the chain being a natural next step.

Figure 14.6 outlines the flowchart of the structure of the on-chip implementation. The similarity to the off-chip implementation structure is evident, but this flowchart features a greater number of blocks. This is owing to the main differences between MATLAB and Arduino as programming

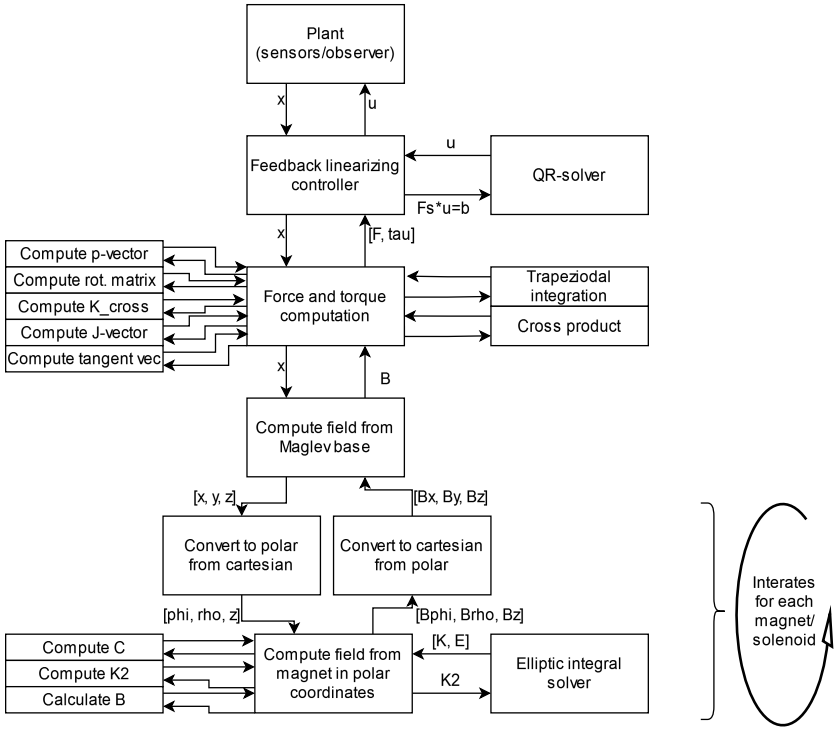


Figure 14.6: Flowchart of the structure of the Arduino implementation for the Teensy

languages. MATLAB is very well suited to matrix calculations and linear algebra, a functionality that must be manually implemented in Arduino through additional functions.

Function adaptations

The MATLAB implementation consists of many integrated functions specific to MATLAB, which is crucial for making the implementation work. This poses the greatest issue for adapting and developing an Arduino alternative for the on-chip approach. With this in mind, the first step of development is to identify these functions and develop Arduino-specific versions of them. The GitHub repository[11] *contains all code relevant for the following adaptation under student_contributions/ code/ PIDKF_master/ Implementation_on_chip/ in the sketch on_chip_Teensy. The repository is also included as an attachment to this thesis.*

[11] https://github.com/Hansolini/Take-home-Maglev-lab

Matrix and vector algebra *One of the first challenges encountered with the adaptation is how MATLAB can inherently deal with vector and matrix algebra. This owing to the fact that it was initially developed as a matrix calculator.*[12] *This functionality is not easy to replicate without additional libraries in Arduino, either acquired or developed separately. For this adaptation, the library BasicLinearAlgebra for Arduino was used, which offers*

[12] Cleve Moler and Jack Little. A history of matlab. *Proc. ACM Program. Lang.*, 4(HOPL), jun 2020. URL `https://doi.org/10.1145/3386331`

basic matrix representation and arithmetics, as well as some more advanced methods such as inversion and transposition.[13] *This library allows for the representation of both matrices and vectors in Arduino and serves as the basis for the whole implementation. However, some methods and operators, such as scalar multiplication and vector dot product, are still not included in this library and must therefore be implemented manually in the code.*

[13] Tom Stewart. Basiclinearalgebra. https://github.com/tomstewart89/BasicLinearAlgebra?tab=MIT-1-ov-file, 2024

ellipke *The first of the MATLAB-specific functions to be adapted to the Arduino implementation is ellipke. This function takes a scalar or a vector of scalars and returns the complete elliptic integral of the first and second kinds, respectively.*[14] *This function is used in the magnetic field computation of each magnet and solenoid and corresponds to the functions K and E in* (10.8).

[14] Inc. The MathWorks. ellipke, b. URL https://se.mathworks.com/help/matlab/ref/ellipke.html

The algorithm MATLAB uses for this calculation is largely based on the Arithmetic-Geometric Mean-process outlined in Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.[15] *The adapted Arduino code closely follows the same structure as the MATLAB function to replicate the same behavior as in the simulator. An example of where the MATLAB function differs from the algorithm proposed by Owen, Abramowitz and Stegun, 1965, is the introduction of an error tolerance, allowing for some inaccuracies and providing some wiggle room with regard to numerical bugs.*

[15] D. B. Owen, Milton Abramowitz, and Irene A. Stegun. Handbook of mathematical functions with formulas, graphs, and mathematical tables. *Technometrics*, 7:78, FEB 1965. ISSN 0040-1706. DOI: 10.2307/1266136. URL https://dx.doi.org/10.2307/1266136

The same tweak was also implemented in the Arduino adaptation to accommodate any numerical deviations that may arise. This differs from the MATLAB function in that the error tolerance is constant, while MATLAB uses a built-in, adaptive tolerance scheme.

trapz *When computing the total force and torque acting on the levitating magnet the main task relates to integrating over the discretized circumference. In the MATLAB implementation, the function trapz is used, which employs the trapezoidal rule to integrate over one or two inputs numerically.*[16]

[16] Inc. The MathWorks. trapz, j. URL https://se.mathworks.com/help/matlab/ref/trapz.html

This might be the easiest to adapt to Arduino, because it only takes up one line of code. What complicates matters is, of course, to adapt it in a way that allows for vectors to be passed as arguments. This is solved by using a for-loop and iterating over all the elements of the vectors to produce an overall answer.

mldivide *The most important piece of code with regard to the controller itself is the way the linear system of equations relating to the input is solved. Since MATLAB is largely based on matrices, custom operators such as *

or the function mldivide exist to handle inline solving of systems of linear equations.

This function uses different methods to solve a given set of equations, i.e. $Ax = b$*, and is very flexible with regard to dimensions and choice of algorithm. Figure 14.7 shows the decision tree used by mldivide when the*

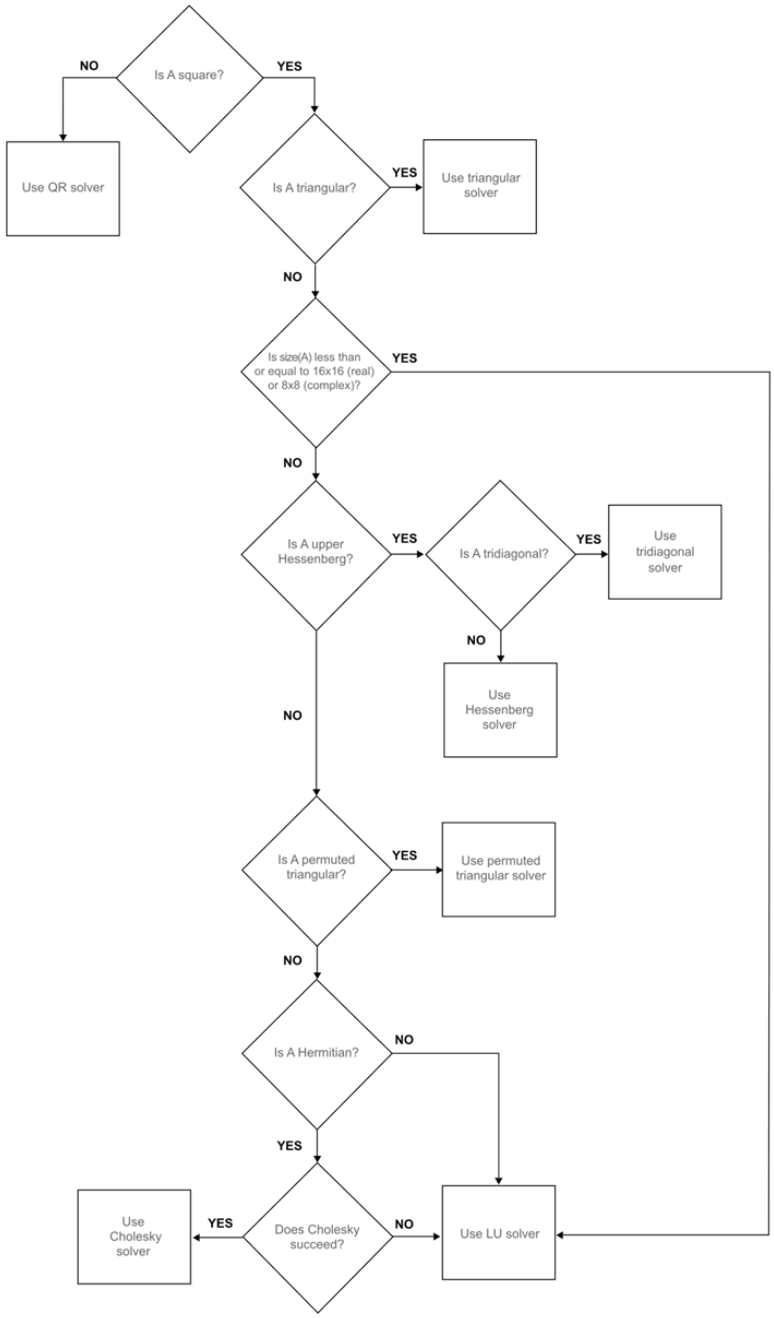


Figure 14.7: Flowchart showing the versatility of the *mldivide* function

function is called with input matrix A. What is apparent from this flowchart is how the implementation of the mldivide function should be handled in the Arduino adaptation.[17]

[17] Inc. The MathWorks. mldivide, c. URL https://se.mathworks.com/help/matlab/ref/mldivide.html

Recall that the controller is given by

```
u = F_s\(h-F_m*I_m-g*K*x)
```

in the MATLAB implementation. Here, the mldivide function is called with the left and right sides of \being A and b, respectively. The dimensions of these parameters are known and constant, 5×4 *for A and* 5×1 *for b, at least for the reduced-state feedback. It is evident that A is not square, and from the flowchart in Figure 14.7, a QR solver is needed.*

The most involved step of implementing the QR solver is choosing a method of decomposing the A matrix into Q and R. The Arduino adaptation includes two methods, namely the Gram-Schmidt and the Householder methods. Both methods involve dividing the A matrix into column vectors and transforming them into perpendicular projections.

Computation time

In the same manner as for the off-chip implementation, the total computation time must be measured for this implementation as well. This is done with the same method as earlier, by placing timers in the code to measure the times of the distinct parts of the computations. In Arduino, this is done by using the micros() function, which returns the number of microseconds that have passed since the start of the program.[18]

[18] Arduino. micros(), 2021. URL https://www.arduino.cc/reference/en/language/functions/time/micros/

Just like earlier, we are interested in two main measurements, one for the force and torque calculations, and one for the controller itself.

Figure 14.8 shows the composition of the two portions we aim to measure the running time of. This constitutes the same parts as the off-chip counterpart in Figure 14.2. The procedure is the same as for the off-chip test, only this time wholly implemented on the Teensy. The code for this test is available under student_contributions/code/ PIDKF_master/Implementation_on_chip/on_chip_Teensy in the GitHub repository[19]*. The repository is also included as an attachment to this thesis.*

[19] https://github.com/Hansolini/Take-home-Maglev-lab

Figure 14.9 showcases a comparison between the two measured running times for both implementation approaches. It is immediately apparent that the off-chip implementation vastly outperforms the on-chip adaptation, even including the transmission delays affiliated with data transfer between MATLAB and the Teensy. With a total running time of ~ 4.2*ms per computation, the Teensy implementation can be expected to support a sampling*

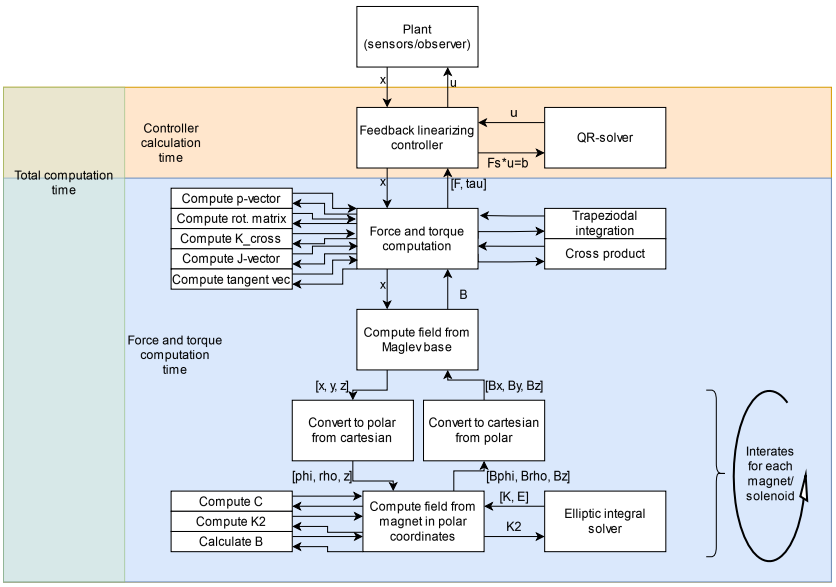


Figure 14.8: Illustration of the different timed portions of the Arduino adaptation

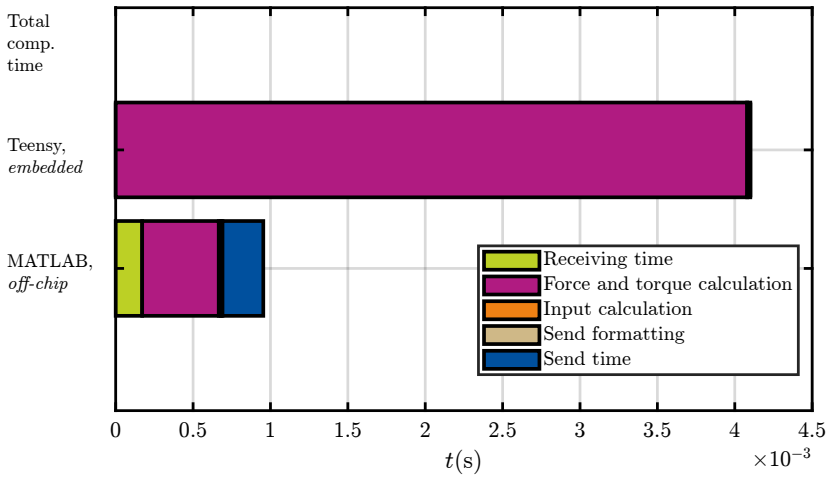


Figure 14.9: Bar graph comparing the running time of the Arduino adaptation to the MATLAB implementation

frequency of $\frac{1}{0.0042} = 238Hz$*. Considering the running time of the off-chip implementation being* $\sim 1ms$*, this approach can support a sampling frequency of* $\frac{1}{0.001} = 1kHz$*, which is vastly faster than the on-chip approach.*

Remarks on implementation

Having now presented two main modes of implementation in detail, a short analysis of their respective suitabilities can be made together with some remarks about the method and potential issues.

On-chip remarks

From the resulting bar graph presented in Figure 14.9, it is clear that the implementation mode of choice should be the off-chip approach. Despite transport delay being larger than first anticipated, it still outperformed the Teensy adaptation. The fact that it was so much better was also surprising, given that pre-compiled code languages tend to have lower running times than interpreted ones.

There can be many reasons for the underperformance of the Arduino code. The main culprit most likely being that the code is far from optimal. The widespread usage of for-loops to handle operations that MATLAB tackles intuitively, i.e. vector dot products, is a major source of computational complexity. A Teensy code built upon a custom-developed and somewhat optimized library for Maggy will most likely have a significantly lower running time than the code tested above. This would drastically improve the suitability of the on-chip approach.

The Arduino code also has some numerical flaws, specifically related to the QR solver. This is the main reason for the implementation of two different methods. As covered briefly in Chapter 1, Gram-Schmidt is not numerically stable, which is the reason for the implementation of the Householder Transformation. We therefore anticipated that the Gram-Schmidt approach would encounter issues, but it seems like the Householder method also suffers from the same flaws.

Specifically, the problem arises when the Q matrix is generated. The first couple of columns are correct, while the last two or three are consistently wrong with what MATLAB calculates. In practice, this means that the developed code for the on-chip implementation cannot be used in its present state. It was decided against spending time correcting this after the initial tests showed that the off-chip implementation performed better.

Off-chip remarks

Since the off-chip approach is evidently the way to go, the next endeavor will consist of trying to minimize the transmission time further. With the measurements from MATLAB showing a drastic decrease in transmission speed in relation to the Teensy's ability to transmit data, looking at probable causes for this allows us to develop some potential remedies for this problem.

A faulty cable could compromise the data transfer, making transmission speed lower than it normally would be. Another cause might be that the cable does not support any of the fastest standards, namely 2.0 (480Mbps) and up, to get the most out of Teensy's native USB transmission speed.

Testing with other cables did not make any difference in the transmission speed to MATLAB. It is also important to ensure that the port used on the receiving device also supports these standards.

As discussed in Part III, Teensy communicates at native USB speeds at up to 480 Mbit/s, and therefore the assumption that the baud rate set in MATLAB does not matter is made. Given that the results obtained were so much worse than the earlier estimates, further investigation is warranted into whether this is correct. Therefore, another transmission speed test was done, this time with a baud rate of 2000000 instead of 1000000. This test also yielded around 55s, therefore confirming the assumption made in the method.

Another avenue to investigate is if the software imposes any limitations. It is evident from Part III that Teensy is not limited on its side, but MATLAB might be. The only way to really test this is to try some other software and compare its performance with MATLAB's. The software of choice for this endeavor is Python because of its easy and intuitive syntax while also offering a wide range of modules for different functionalities, such as serial communication through pySerial.[20,21]

[20] Chris Liechti. pyserial. URL https://pyserial.readthedocs.io/en/latest/pyserial.html

[21] Simon Yuill and Harry Halpin. Python. *Python releases for windows*, 24, 2021

A simple Python code allows us to conduct a test like with MATLAB to deduce a transmission speed between Python and Teensy. By following the same methodology as before, the relation between the time Teensy uses to print 500000 lines and the time Python uses to receive it can be established. The test revealed that Python used around 33 seconds to receive all the data, which is a little over half of what MATLAB did. This means that the transmission speed between Python and Teensy is almost double that of MATLAB. The python code for this test is located at student_contributions/code/PIDKF_master/Implementation_off_chip/Transmission_Speed_test in the GitHub repository[22]*. The repository is also included as an attachment to this thesis.*

[22] https://github.com/Hansolini/Take-home-Maglev-lab

This serves to show that the software choice made for the off-chip implementation might not be optimal. Other programming languages have risen to match MATLAB and its computing properties, especially Python with its flexible NumPy library.[23,24] *Implementation on different platforms than MATLAB might therefore provide a better base for stable implementation.*

[23] David Ascher, Paul F. Dubois, Konrad Hinsen, Jim Hugunin, Travis Oliphant, et al. Numerical python, 2001

[24] K. Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science & Engineering*, 13(2):9–12, 2011. DOI: 10.1109/MCSE.2011.36

Simulator considerations

The results from the implementation studies also gave some insight into how to improve the simulator used for the stability analysis. The analysis results presented in Chapter 13 were based on an entirely continuous model

for the entire system, both dynamics and controller. The reasoning behind this approach was basically that at the point of the analyses being made, we didn't know enough details about how an actual implementation would influence the system. By exploring the implementation opportunities, we now know that the controller will have to be sampled in some way. With the knowledge gathered from the off-chip implementation study, we can now propose a somewhat optimal sampling time based on the delays introduced by calculation and data transmission.

By placing a zero-order hold block in the simulator's Simulink diagram, we can simulate a sampled version of the system measurements. This will allow for more accurate simulations and more accurate approximations of the regions of convergence. Further adaptations can also be made to the simulator, like imposing a specific delay between the calculation of the input and the dynamics themselves to simulate the transmission time for input calculation. All of this will allow us to examine the feasibility of the controller further and, more realistically, propose tuning parameters that will work on the physical system. These proposed changes are detailed in blue in Figure 14.10.

Zero-order hold: a Simulink block that holds its input constant for a specified period, simulating a sampled signal, The MathWorks, Inc., a.

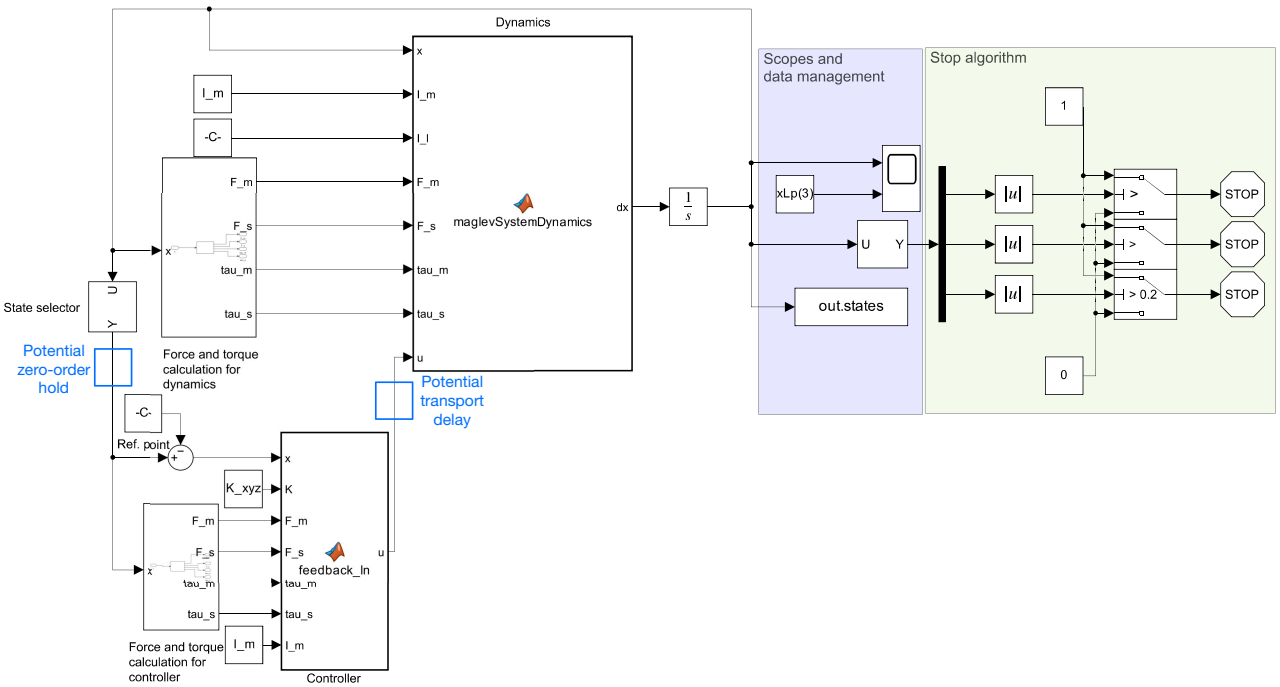


Figure 14.10: Simulink diagram with proposed changes

Chapter 15

Concluding remarks: Stability and controller implementation

The two preceding chapters detail work done on the system in order to stabilize it. The analyses presented were conducted through the methods explained, and serve to prove that the system can be controlled. The analyses culminated in the development of a feedback linearizing controller with a pole placement control scheme. The use of a simulator developed in MATLAB and Simulink was used to test different controller schemes, and underline their properties and performances.

With regard to the feedback linearization control scheme, analysis through simulation revealed the impact different limitation has on its performance. It was found to generally handle limitations much better than the LQR controller, especially when a transport delay was introduced to the system. The controller also performed reasonably well when constraints were put on the input, but for a physical implementation to be feasible, these constraints must be tightened further.

Different controller implementation methods were presented, along with advantages and disadvantages. Tests were developed to assess their performances in relation to each other, as well as with regard to controller stability. These tests revealed the off-chip approach to be best suited, even though some enhancements must be made to make it function fully.

This concludes the inquiry into the stabilization of the system. It is the basis of further development of Maggy from a control perspective and can be used to formulate control-related lab assignments, as well as assignments related to controller implementation.

Part V

Magrobatics & Beyond

Introduction

In this part, we discuss the future of Maggy. We discuss potential use cases and novel modes of operation, challenges in the current setup — both control-related and mechanical —, planned and potential changes that can be made to Maggy, and the way forward. Aside from outlining the future of our development, we hope that this will inspire others to innovate on Maggy, allowing for more exciting and educational take-home labs.

Chapter 16

The future of Maggy

Further work: Stability and controller implementation

Throughout developing the controller and investigating implementation approaches, we have acquired information about the system that lets us propose further improvements to the controller and the methodology. This is boiled down to the endeavors presented in the following paragraphs.

Simulator adaptations *To more closely resemble the system, we can introduce some adaptations in the simulator used for stability analysis. A possible set of changes has already been described and is detailed in Figure 14.10. Other approaches to liken the simulator to the physical system should also be investigated, such as switching any transport delay blocks with a unit delay block.*

Input constraints *The biggest obstacle to successfully implementing the controller on Maggy, not counting the need for a state estimator, is limiting the input sufficiently. The thesis proposes sets of poles that stabilize the system with input constraints down to* ± 50*, but this must be brought down further. The primary mode for adapting the system to tighter input constraints presented in this thesis is the adaptation of the poles. The poles themselves are acquired through trial and error while the system's response to different sets of poles is observed. If we can deduce a more effective method for placing more proportional, but still stable, poles of the system, the procedure of constraint tightening would become easier.*

Off-chip software choice *As presented in the implementation study of the off-chip approach, the choice of MATLAB as the software to handle the data might not be optimal. The discussion following this analysis outlined a scenario in which Python was used instead, which seemed to increase data transmission speed in comparison to MATLAB. Therefore, an interesting endeavor is to investigate whether implementation through other software*

further increases data transmission speed and whether it is possible to implement the calculation algorithms needed for the controller.

On-chip code optimization *Another avenue to explore concerning implementation is to optimize the code for the on-chip implementation. As discussed in Chapter 14, a specialized library for Maggy that can be used on the Teensy can go a long way in limiting computational times. Of course, this is more time-consuming than trying to increase data transmission speeds for the off-chip approach, but it might be an interesting challenge for someone more proficient in programming than myself.*

Magrobatics

The concept of Magrobatics (combining acrobatics and magnetism) is a proposal for a new mode of operation that we think could pave the way for new exciting and educational problems with Maggy. The inspiration for this concept is the realization that, for certain configurations of permanent magnets, there may be more than one equilibrium point for the levitating magnet to be stabilized in. Magrobatics essentially entails designing controllers that are capable of moving the levitating magnet between these equilibrium points, using, for instance, switched linear controllers with some clever reference design, or advanced model-based control methods. One particularly novel and interesting is to make the levitating magnet flip 180 degrees in the air, as there are configurations where the magnet could levitate stably in both orientations. This might be extremely challenging due to the limited power of the electromagnets, but would be very interesting, academically and educationally, if possible

Bibliography

Handling offsets and trends in data. URL https://www.mathworks.com/help/ident/ug/handling-offsets-and-trends-in-data.html.

Vaf. URL https://www.dcsc.tudelft.nl/~jwvanwingerden/lti/doc/html/vaf.html.

Principles of electronic communication systems. *McGraw-Hill, a business unit of The McGraw-Hill Companies, Inc., 2014.*

Hirotugu Akaike. Canonical correlation analysis of time series and the use of an. 1976.

Trond Andresen, Jens G. Balchen, and Bjarne A. Foss. Reguleringsteknikk. *Institutt for teknisk kybernetikk, NTNU, 2016. ISBN 9788278422021.*

Arduino. char, 2019. URL https://www.arduino.cc/reference/en/language/variables/data-types/char/.

Arduino. micros(), 2021. URL https://www.arduino.cc/reference/en/language/functions/time/micros/.

Arduino. Serial.write(), 2023. URL https://www.arduino.cc/reference/en/language/functions/communication/serial/write/.

David Ascher, Paul F. Dubois, Konrad Hinsen, Jim Hugunin, Travis Oliphant, et al. Numerical python, 2001.

Roberto Bassani. Earnshaw (1805–1888) and passive magnetic levitation. Meccanica, *41:375–389, 2006.*

Vangie Beal. Definitions: Serial port, 2021. URL https://www.webopedia.com/definitions/serial-port/.

Rakesh P. Borase, D. K. Maghade, S. Y. Sondkar, and S.N. Pawar. A review of pid control, tuning methods and applications. International Journal of Dynamics and Control, *9:818–827, 2020.* DOI: *https://doi.org/10.1007/s40435-020-00665-4.*

Martin Brønstad, Jørn Olav Deila, Jørgen Dyrskog, and Markus Langklopp. Magnetic levitation system: Design, prototyping and testing of a digital pid-controller. Bachelor's thesis, Norwegian University of Science and Technology, Trondheim, 2022.

Guanrong Chen. Stability of nonlinear systems. In Kai Chang, editor, Encyclopedia of RF and Microwave Engineering, *pages 4881–4896. Wiley, New York, 2004. URL* https://onlinelibrary.wiley.com/doi/10.1002/0471654507.eme413. *Accessed: 2024-05-28.*

David Di Ruscio. Subspace system identification: Theory and applications. Lecture notes, Telemark University College, Porsgrunn, Norway, *1997.*

Hans Alvar Engmark and Kiet Tuan Hoang. Modeling and control of a magnetic levitation platform. IFAC-PapersOnLine, *56(2):7276–7281, 2023. ISSN 2405-8963.* DOI: *https://doi.org/10.1016/j.ifacol.2023.10.338. URL* https://www.sciencedirect.com/science/article/pii/S240589632300705X. *22nd IFAC World Congress.*

Pål Ivar Delphin Kværnø Fosmo. Control of a magnetic levitation system using feedback linearization. 2023.

Walter Gander. Algorithms for the qr decomposition. Res. Rep, *80(02): 1251–1268, 1980.*

Gene H. Golub and Charles F. Van Loan. Matrix computations. *JHU press, 2013.*

Migdonio A. González and Dorindo E. Cárdenas. Analytical expressions for the magnetic field generated by a circular arc filament carrying a direct current. IEEE Access, *9:7483–7495, 2021.* DOI: *10.1109/ACCESS.2020.3044871.*

Sverre Graffer. State estimation of a maglev system with the luenberger observer. 2023.

I. S. Grant. Electromagnetism. *Wiley, 1990. ISBN 978-0-471-92712-9.*

Maki K Habib and Samuel A Ayankoso. Data-driven modeling: Concept, techniques, challenges and future prospects.

Joao P. Hespanha. Linear systems theory. *Princeton university press, 2018.*

Hoshinchu. Air bonsai concept. http://hoshinchu.com/, *2024. Accessed: 2024-06-02.*

Tohru Katayama et al. Subspace methods for system identification, *volume 1. Springer, 2005.*

Hassan K. Khalil. Nonlinear Systems. *Prentice Hall, 2002. ISBN 0130673897.*

Carl Knospe. Pid control. IEEE Control Systems Magazine, *26(1):30–31, 2006.* DOI: *10.1109/MCS.2006.1580151.*

Arthur J. Krener and Kayo Ide. Measures of unobservability. In Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference, *pages 6401–6406, 2009.* DOI: *10.1109/CDC.2009.5400067.*

Kenneth Lange. Singular value decomposition. Numerical analysis for statisticians, *pages 129–142, 2010.*

Chris Liechti. pyserial. URL `https://pyserial.readthedocs.io/en/latest/pyserial.html`.

Lennart Ljung. System identification toolbox: User's guide. *Citeseer, 1995.*

K. Jarrod Millman and Michael Aivazis. Python for scientists and engineers. Computing in Science & Engineering, *13(2):9–12, 2011.* DOI: *10.1109/MCSE.2011.36.*

Cleve Moler and Jack Little. A history of matlab. Proc. ACM Program. Lang., *4(HOPL), jun 2020. URL* `https://doi.org/10.1145/3386331`.

Geir Werner Nilsen. Topics in open and closed loop subspace system identification: finite data-based methods. 2005.

D. B. Owen, Milton Abramowitz, and Irene A. Stegun. Handbook of mathematical functions with formulas, graphs, and mathematical tables. Technometrics, *7:78, FEB 1965. ISSN 0040-1706.* DOI: *10.2307/1266136. URL* `https://dx.doi.org/10.2307/1266136`.

PJRC. Tutorial 1: Software setup, a. URL `https://www.pjrc.com/teensy/tutorial.html`.

PJRC. Teensyduino: Using usb serial communication, b. URL `https://www.pjrc.com/teensy/td_serial.html`.

S Joe Qin. An overview of subspace identification. Computers & chemical engineering, *30(10-12):1502–1513, 2006.*

R. Ravaud, G. Lemarquand, S. Babic, V. Lemarquand, and C. Akyel. Cylindrical magnets and coils: Fields, forces, and inductances. IEEE Transactions on Magnetics, *46(9):3585–3590, 2010.* DOI: *10.1109/TMAG.2010.2049026.*

John Ringwood. Multivariable control using the singular value decomposition in steel rolling with quantitative robustness assessment. Control Engineering Practice, *3(4):495–503, 1995.*

Arturo Rojas-Moreno and Cristian Cuevas-Condor. Pd and pid control of a maglev system an experimental comparative study. In 2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON), *pages 1–4, 2017.* DOI: *10.1109/INTERCON.2017.8079678.*

Matthew N. O. Sadiku. Elements of Electromagnetics. *Oxford University Press, USA, 2001. ISBN 019513477X.*

Shankar Sastry. Nonlinear Systems: Analysis, Stability, and Control, *chapter 5. Springer New York, 1999. ISBN 978-1-4757-3108-8.*

Hannes Siebeneicher. Universal asynchronous receiver-transmitter (uart), 2023. URL https://docs.arduino.cc/learn/communication/uart/.

J. E. Slotine and W. Li. Applied Nonlinear Control. *Prentice Hall, Inc, 1991.*

Tom Stewart. Basiclinearalgebra. https://github.com/tomstewart89/BasicLinearAlgebra?tab=MIT-1-ov-file, *2024.*

Paul Stoffregen. Usb-serial-print-speed-test. https://github.com/PaulStoffregen/USB-Serial-Print-Speed-Test/tree/master, *2019.*

Inc. The MathWorks. Zero-order hold, a. URL https://se.mathworks.com/help/simulink/slref/zeroorderhold.html.

Inc. The MathWorks. ellipke, b. URL https://se.mathworks.com/help/matlab/ref/ellipke.html.

Inc. The MathWorks. mldivide, c. URL https://se.mathworks.com/help/matlab/ref/mldivide.html.

Inc. The MathWorks. place, d. URL https://se.mathworks.com/help/control/ref/place.html.

Inc. The MathWorks. Write and read serial port data, e. URL https://se.mathworks.com/help/matlab/import_export/write-and-read-serial-port-data.html.

Inc. The MathWorks. serialport, f. URL https://se.mathworks.com/help/matlab/ref/serialport.html.

Inc. The MathWorks. svd, g. URL https://se.mathworks.com/help/matlab/ref/double.svd.html.

Inc. The MathWorks. tic, h. URL https://se.mathworks.com/help/matlab/ref/tic.html.

Inc. The MathWorks. toc, i. URL https://se.mathworks.com/help/matlab/ref/toc.html.

Inc. The MathWorks. trapz, j. URL https://se.mathworks.com/help/matlab/ref/trapz.html.

Rudi Uswarman, Adha I. Cahyadi, and Oyas Wahyunggoro. Control of a magnetic levitation system using feedback linearization. In 2013 International Conference on Computer, Control, Informatics and Its Applications (IC3INA), *pages 95–98, 2013.* DOI: *10.1109/IC3INA.2013.6819156.*

Ramon Vilanova and Antonio Visioli. PID control in the Third Millennium : lessons learned and new approaches. *2012.*

Simon Yuill and Harry Halpin. Python. Python releases for windows, *24, 2021.*

Qian Zhang and Bingqiang Li. Feedback linearization pid control for electro-hydrostatic actuator. In 2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), *pages 358–361, 2011.* DOI: *10.1109/AIMSEC.2011.6010249.*

NTNU

Kunnskap for en bedre verden