Systems and Control Library
Development

B. Mrkajić
CST 2011.058

Master’s thesis

Coach(es):  dr.ir. M.J.G. van de Molengraft
            ir. J.J.M. Lunenburg

Supervisor:  prof.dr.ir. M. Steinbuch

Committee:   prof.dr.ir. M. Steinbuch
             dr.ir. M.J.G. van de Molengraft
             dr. H. Bruyninckx†
             ir. J.J.M. Lunenburg

† KU Leuven Belgium

Eindhoven, August, 2011
Preface

This report describes the results of my graduation project, named "Systems and Control Library Development", of Master's track Systems and Control at Eindhoven University of Technology (TU/e). The work has been carried out in the Control Systems Technology group at the department of Mechanical Engineering at TU/e, Netherlands in the period from January to August 2011.

Over 8 months ago this project started as a huge leap into unknown territory for me and it was a great challenge of exploring a new field of interest. During my graduation period thus I had contact with people from numerous fields of expertise and background, with whose help I managed to conduct the project. Herewith I would like to show my gratitude to the persons who have helped me with my research during the graduation period.

In the first place, I would like to thank to my coaches, professor dr. ir. René van de Molengraft and Janno Lunenburg for their support, advising and freedom in the choice of the course of the project. Secondly, I want to express gratitude to professor dr. Herman Bruyninckx from K.U.Leuven, Belgium for discussions and inspiring ideas regarding several parts of the project. Furthermore, I want to thank my colleagues from Dynamics and Control Technology laboratory for practical and technical help. Special thanks I owe to Sava Marinkov, for his patient listening, endless discussions and rousing suggestions. Finally, I would like to thank my programme supervisor, professor dr. ir. Maarten Steinbuch for help in arranging this research and for his academic advising throughout my education at TU/e.

Last but not least, I would like to show gratitude to my parents Vera and Rajko, brother Veljko and friends for their great support during the whole course of my studies.

Boris Mrkajić
Summary

Over the past decades control engineers have developed various tools in order to make an assortment of control functionalities possible and easy to use. This holds for the functionalities that have to run in real-time, like PID or adaptive control, as well as for the control design tools, like systems identification, pole placement or loop-shaping. Nevertheless, these tools are generally expensive, incomplete and/or solely oriented toward specific applications. Hence, the idea of the Systems and Control Library emerged and this project was initiated. The main goal of the project is to establish a reusable, comprehensive, versatile, open source library, which would consist of all the needed control-related tools and functionalities in such a manner that it can be used in different environments and by different platforms, both commercial and non-commercial.

Within this report, the whole software development process that was carried out is described. This approach was crucial for accomplishing the set goals and it is hence elaborated in details. Firstly, the library requirements are specified, i.e., tools that belong to the designed library and the ones that do not are determined. A lack of the right tools in the library would hamper its usability and thus its receptivity. On the other hand, by feckless and excessive choice of the functionalities, the library would gain quantity, but lose on its quality. Additionally, various aspects that are required to be considered during the software design phase are analyzed, like reusability, compatibility or modularity of the software. Achieving these aspects adds value to the library in terms of usability, but also makes further development easier.

Next, the decisions regarding software architecture are presented. From an extensive range of possibilities, the one that could fulfill the set requirements and that has proven its efficiency in practice is chosen. Moreover, the choice of programming language is made, based on a minute literature survey. Software licensing, copyright information and documentation that are needed for the library are presented as well in this section. All the mentioned decisions that were made are elaborated and they should enable a clear overview of the library, helping both developers and users to understand it and finally use it.

The establishment of the library software basis was followed by the second part of the project, which is the integration of a segment of the library, together with the required theory. This integration comprises only the fundamental core of the library, expressed through digital filters, together with a segment of the tools needed for the planned initial application, which is control of an omnidirectional mobile platform. Digital filter definition is given, followed by filter realization techniques, which are analyzed meticulously due to the importance of the choice that influences the final performance. Moreover, a number of filter discretization methods are considered and evaluated. Lastly, the mobile platform control
part is composed of definitions of general wheeled platforms and their classification with respect to different criterions. This section of the project is concluded by the derivation of solvers which enable decoupled control, and their implementation.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>i</td>
</tr>
<tr>
<td>Summary</td>
<td>iii</td>
</tr>
<tr>
<td>Contents</td>
<td>v</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation and Background</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Objectives</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Outline of the Report</td>
<td>4</td>
</tr>
<tr>
<td>2 Software Development Process</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Definition of a Software Library</td>
<td>5</td>
</tr>
<tr>
<td>2.1.1 Static libraries</td>
<td>6</td>
</tr>
<tr>
<td>2.1.2 Shared libraries</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Survey of Currently Available Tools</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Systems and Control Library Requirements</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 Considered Aspects</td>
<td>11</td>
</tr>
<tr>
<td>2.3.2 Software Requirements Specification</td>
<td>12</td>
</tr>
<tr>
<td>2.4 Software Design</td>
<td>14</td>
</tr>
<tr>
<td>2.4.1 5Cs Approach</td>
<td>15</td>
</tr>
<tr>
<td>2.4.2 Selection of Library Type</td>
<td>18</td>
</tr>
<tr>
<td>2.4.3 Programming Language Choice</td>
<td>19</td>
</tr>
<tr>
<td>2.4.4 Component Oriented Programming</td>
<td>20</td>
</tr>
<tr>
<td>2.5 A Systems and Control Library Component</td>
<td>21</td>
</tr>
<tr>
<td>2.6 Licensing and Copyright</td>
<td>23</td>
</tr>
<tr>
<td>2.6.1 Licensing</td>
<td>23</td>
</tr>
<tr>
<td>2.6.2 Copyright</td>
<td>23</td>
</tr>
<tr>
<td>2.7 Documentation</td>
<td>24</td>
</tr>
<tr>
<td>3 Digital Filters</td>
<td>25</td>
</tr>
<tr>
<td>3.1 Filter Definition</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Filter Realization</td>
<td>27</td>
</tr>
<tr>
<td>3.2.1 The Four Direct Forms</td>
<td>27</td>
</tr>
<tr>
<td>3.2.2 High Order Filter Decomposition</td>
<td>31</td>
</tr>
<tr>
<td>3.3 Filter Discretization</td>
<td>32</td>
</tr>
<tr>
<td>3.3.1 Survey of Discretization Methods</td>
<td>32</td>
</tr>
<tr>
<td>3.3.2 Standard Digital Filters</td>
<td>38</td>
</tr>
<tr>
<td>3.4 Filter Utilization Example</td>
<td>53</td>
</tr>
</tbody>
</table>
## CONTENTS

### 4 Kinematic Models of Wheeled Mobile Robots
- 4.1 Robot Posture .................................................. 58
- 4.2 Description of Wheels ....................................... 59
  - 4.2.1 Conventional Wheels .................................. 59
  - 4.2.2 Swedish Wheels ......................................... 61
- 4.3 Definition of Kinematic Families ......................... 62
  - 4.3.1 Restrictions to the Robot Mobility .................. 62
  - 4.3.2 Degree of Mobility and Degree of Steerability ... 64
  - 4.3.3 Classification ........................................... 65
- 4.4 Solvers for Kinematic Families ......................... 67
  - 4.4.1 Fixed Wheel Jacobian Matrix Calculation .......... 68
  - 4.4.2 Time-Varying Wheel Jacobian Matrix Calculation .. 71

### 5 Conclusions and recommendations
- 5.1 Conclusions .................................................. 73
- 5.2 Recommendations ........................................... 75

## Appendix

### A Generic Formula for $s$-domain to $z$-domain Transformation
- .................................................. 77

### B PID Anti-Windup
- B.1 Set Point Limitation ..................................... 79
- B.2 Conditional Integration .................................... 79
- B.3 Back Calculation Method .................................. 80

### C Detailed Derivation of Standard Digital Filters’ Transfer Functions
- C.1 First Order Lowpass ........................................ 83
- C.2 Second Order Lowpass ...................................... 85
- C.3 Weak Integrator ............................................. 88
- C.4 Lead Lag .................................................... 89
- C.5 Skewed Notch ............................................... 92
- C.6 PD .......................................................... 94
- C.7 Improved PD ................................................ 96
- C.8 PID .......................................................... 98
- C.9 Improved PID ................................................. 100

### D Examples of the Code
- D.1 Digital Filters Example .................................... 105
- D.2 Wheeled Mobile Platforms Decoupling Example .... 110

### E Test Results of the General Matrix Inverse Calculation
- .................................................. 115

## Bibliography
- .................................................. 117
Chapter 1

Introduction

1.1 Motivation and Background

In the last few years the Eindhoven University of Technology (TU/e) has been actively involved in many robotics projects. Starting with participation in RoboCup, an international research initiative [50], followed by founding the RoboEarth project [51], SOFIE project [7], etc. Apart from being well-known in high-performance motion control, the Department of Mechanical Engineering of TU/e is emerging as a relevant institution in the field of robotics as well.

Robotics is a field of modern technology that crosses traditional engineering boundaries, requiring and combining knowledge from numerous fields. Essentially, the goal of robotics is to enable machines to execute certain tasks. In order to execute them with a robot, the mechanical structure of it requires to be first designed, but finally also controlled. The control of a robot involves three distinct phases: perception, processing and action. This is also known as the robotic paradigm [5], including three primitives: sense, plan and act (see Figure 1.1). Sensors give information about the environment or the robot itself, e.g., the position and velocity of its joints. This information is further processed to calculate the appropriate signals for the actuators, which move the mechanics.

![Figure 1.1: Robotic paradigm](image)

Starting all the mentioned projects, one comes across the three primitives in the paradigm. For the sake of making them possible, both hardware and software requirements need to be satisfied. Just like human needs the body and the soul to be able to live, robots need hardware and software to be able to do their function. The former serves as "body" by which they can contact with their environment, while the latter serves as "soul" where the program...
and command stored. In other words, the hardware fulfills the needs of perception (sensing) and action, while software takes care of processing (planning). Hence, the presented robotic paradigm can be directly translated to control block diagram structure in Figure 1.2. The analogy is as follows: perception is enabled using robot sensors, action using robot actuators and finally processing and planning using controllers.

![Figure 1.2: Standard control block diagram in robotics](image1)

This concept can be further generalized since it does not hold only for robotics, but all controlled systems. In general, the control block diagram can be depicted as in Figure 2.11. The plant is any system to be controller (e.g., the robot), sensors are devices that measure physical quantities (e.g., positions, temperatures, etc.) and enable the controller to be aware of the current status of the plant. Using that information and the desired behavior of the system, the controller decides on new control inputs to the plant, i.e., to the plant actuators.

![Figure 1.3: General control block diagram](image2)

Having in mind everything stated above, a natural question emerges: Is there a tool that meets all the requirements of first robotic paradigm, but also needs of general control structure? Over the past decades control engineers have developed various tools in order to make an assortment of control functionalities possible and easy to use. This holds for the functionalities that have to run in real-time, like PID or adaptive control, as well as for the control design tools, like systems identification, pole placement or loop-shaping. Survey of the currently available tools will be elaborated in Chapter 2. Nevertheless, these tools are generally expensive, incomplete and/or solely oriented toward specific applications. Mainly, they are closed and commercial projects. This fact hampers and slows down the course of development.

Hence, the idea of the Systems and Control Library evolved and this project was initiated. The main goal of the project is to establish a reusable, comprehensive, versatile, open source library, which would consist of all the needed control-related tools and functionalities in such a manner that it is utilizable by different environments and platforms. In
other words, all the functionalities that could be integrated segments of the control block diagrams are supposed to be gathered in one library, widely available for both commercial and non-commercial usage. The explicit objectives will be elaborated in the following section.

1.2 Objectives

As it has been mentioned in the previous section, the need for a library which would consist of all the required control-related tools and functionalities, in such a manner that it is utilizable by different environments and platforms, appears. The goal of this project is exactly that, defining the Systems and Control (S&C) Library by setting a firm software basis and describing all its requirements, which would consist of functionalities that have to run in real-time, as well as for the control design tools. The problem statement of this project is explicitly formulated as:

*Develop and set a firm and clear software basis of an open source library which comprises all the functionalities and tools needed in the field of systems and control.*

In order to have this objective met, the library needs to fulfill the following features:

- **Reusable**
  The property that enables a segment of software to be used again to add new functionalities with slight or no modification. It reduces implementation time, increase the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required.

- **Expandable**
  This feature reflects the possibility of a software project to be expanded by new functionalities. It gives opportunity for other users to contribute to a tool and add a value to it.

- **Compatible**
  The software is said to be compatible, *e.g.*, if it can run on different operating systems or if it can operate with other products and with different environments. Additionally, a piece of software should be backward-compatible with its older version.

- **Fast**
  The library must be able to support real-time utilization, algorithms must be carefully selected and feasible. Special attention should be addressed to avoidance of dynamic memory allocation, which hampers real-time usage.

- **Well-documented**
  Consistent, thorough, simple and clear description of each feature of the library is needed in order to assist the user in realizing these features. Comments in the software should be written such that an auto-generate tool can used for this purpose. This is appealing since it yields a relatively fixed structure of the documentation, *i.e.*, keeps it consistent.

On the other side, this report should represent the performed work on this project, but as well draw a clear picture of the following:
Chapter 1. Introduction

- What is already included in the library and what should be added in the future.
- Which software design decisions were made and why they were made.
- How the library is organized and how it can be enhanced.

All this will not only yield a ready-to-use software, but also enable future contributions easy and fast.

1.3 Outline of the Report

This report is organized as follows. In Chapter 2 the whole software development process that is carried out is presented. Firstly, the library requirements are specified, i.e., tools that belong to the designed library and the ones that are excessive, followed by analysis of the aspects that are to be considered in software design. Next, the design of the software and the decisions regarding it are presented. This comprises the so-called 5Cs concept and its elaboration in the context of Systems and Control Library.

Furthermore, the integration of digital filters is addressed in Chapter 3, together with the required theory and an example. This is followed by the definition and classification of kinematic models of wheeled mobile robots in Chapter 4. Also, solvers for some of the kinematic families are derived and implementation decisions that were made are explained.

Finally, Chapter 5 gives conclusions and recommendations for future work on the topic of the Systems and Control Library.
Chapter 2

Software Development Process

In this chapter, the beginning steps in the library development structure are presented and explained. First, the definition of a general library is given, together with the reasons for its existence. Next, an overview of the presently available tools and frameworks is laid out. This is followed by the requirements for the Systems and Control (S&C) Library, explaining which functionalities should be present and which should not, along with the argumentation. Additionally, various aspects that are required to be considered during the software design phase are analyzed. Lastly, the software design approach and the decisions made regarding it are elaborated. It is important to make the following remark here: this work is followed by the code integration, which is elucidated in subsequent chapters. Integration of the code is executed in several iterations, with the evaluation of the requirements at the end of each one of them. Evaluations were used to assess whether the requirements are achieved, which implied the corresponding planning for the next iteration.

2.1 Definition of a Software Library

A software library could be defined as a collection of resources used to develop software \cite{56}. These may include various classes, values or type specifications. In other words, a library consists of code and data that supply independent programs with services, which generally allows code and data sharing and modification, in a modular fashion. Libraries are usually not executables or stand-alone programs, but in some cases they can be. However, this is not common. Libraries and applications that use them make references known as links between each other through the process known as linking, which is typically done by a linker \cite{34}. Couple of examples are C++ Standard Library \cite{1}, Eigen \cite{26}, Boost \cite{11}, etc. C++ Standard Library is a collection of classes and functions, which are written in the core language and it is integrated in C++ programming language by default. It provides a number of generic containers, functions to utilize and manipulate these containers, function objects, generic strings and streams, and commonly used functions for tasks such as finding the square root of a number. The Eigen library is a stand-alone, unified, matrix library written in C++. It is also a template library, which consists of the tools used in linear algebra. In other words, tools used for handling matrices and vectors. Moreover, it comprises a number of numerical
solvers and related algorithms. The Boost is rather a project, however it represents a set of free portable C++ source libraries. The range of these libraries is wide, from general-purpose libraries, to operating system abstractions and libraries primarily aimed at other library developers and advanced C++ users. Boost also suggest an interesting concept – the majority of libraries are header based libraries, consisting of inline functions and templates, and as such do not need to be built in advance of their usage, which can be practical.

There are several types of library classes, out of which the most important ones are static and shared libraries.

### 2.1.1 Static libraries

Initially, only static libraries existed. A static library or archive, as it is also called, consists of a set of routines which are copied into a target application by the compiler or linker, producing object files and a stand-alone executable file. This process, and the resulting stand-alone executable file, are known as a static build of the target application, see Figure 2.1.

The static linker is responsible for resolving all of the unresolved addresses into fixed or relocatable addresses. This is done by loading all the application code and libraries into actual run-time memory locations, which can sometimes take even more time than the compilation process and must be performed each time when any of the modules is recompiled.

A static linker may only work on specific types of object files, and thus requires specific, compatible types of libraries. Collecting object files into a static library may ease their distribution and encourage their use. A client, either a program or a library subroutine, accesses a library object by referencing just its name. The linking process resolves references by searching the libraries in the order given.

Some programming languages may use a feature called "smart-linking" where the linker is aware of the compiler or it is integrated with it, such that the linker "knows" how external references are used, and code in a library that is never actually used, even though internally referenced, can be discarded from the compiled application. For example, a program
that only uses integers for arithmetic, or does no arithmetic operations at all, can exclude the floating-point library routines. This smart-linking feature can lead to reduced memory usage and smaller application file sizes.

Most programming languages have an integrated standard library, although custom made libraries are naturally possible as well. An example is aforementioned C++ Standard Library.

2.1.2 Shared libraries

Shared linking involves loading the library subroutines into an application program at load-time or run-time, rather than linking them in at compile time. Only a minimum amount of work is done at compile time by the dynamic linker, which only records what library routines the program needs and the index names or numbers of the routines in the library, see Figure 2.2. At the appropriate time the loader adds the relevant data to the process’s memory space. This implies that, using shared libraries, applications can automatically take advantage of improvement in the library, because their link to the libraries is dynamic, not static. In other words, the functionality of the client applications can be improved and extended without requiring application developers to recompile the applications. This fact makes shared libraries most used type of a library.

Moreover, shared libraries by definition offer a form of sharing, allowing the same library to be used by multiple programs at the same time, which is not the case with static libraries. Library sharing can refer to sharing code located on disk by unrelated programs, but also to sharing of code in memory, when programs execute the same physical page of the Random-Access Memory (RAM), mapped into different address spaces. This is archived with position-independent code [24].
2.2 Survey of Currently Available Tools

Knowing what a general software library stands for, a question about the tools that are already available in the field of systems and control arises. Also, their advantages and disadvantages should be analyzed, yielding the reasoning and verifying the motivation for the initiation of this project. This section gives an overview of currently available tools, together with this analysis and drawn conclusions.

As it has been stated in the introduction, nowadays numerous tools are available for an assortment of control functionalities. These functionalities vary from the ones that need to run in real-time, like PID or adaptive control, to functionalities for the control design tools, like system identification, optimization, pole placement, loop-shaping, etc. Nevertheless, these tools are generally commercial, i.e., expensive, rather incomplete, solely oriented toward specific applications and thus hardly adaptable and reusable.

One of the most widely present frameworks is MATLAB™. It is a numerical computing, interactive environment, which allows matrix manipulations, plotting of functions and data, implementation of algorithms and creation of user interfaces. It can be used in a wide range of applications, including signal and image processing, communications, but also control design, test and measurement. Various add-on toolboxes, which represent collections of special-purpose functions, extend the environment to solve particular classes of problems in these application areas. For instance, control system, system identification, robotics or optimization toolboxes are available. It is indeed used in industry, however typically only in the control design phase. Then, when it comes to the implementation, most of the companies develop their own tools, due to the high costs of the software. This new software development costs a lot of engineering hours and is hence unwanted.

In parallel, there is Simulink™, which is an environment for multidomain simulation and model-based design for dynamic and embedded systems. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate,
implement, and test a variety of time-varying systems, including communications, controls, signal processing, etc. A snapshot of Simulink™ library browser is given in Figure 2.3. However, these environments are commercial and thus they do not fit into the open source concept. The open source model comprises concurrent yet different agendas and differing approaches in production, unlike more centralized models of development such as those typically used in commercial software companies. A main principle and practice of open source software development is peer production by barter and collaboration, with the end-product, source-material and documentation available to the public at no cost. This drives the development to be tremendously faster and more efficient, allowing anybody to contribute.

Even though, there are some contributions to, for example, Simulink™, like DCTOOLS© or ShapeIt© developed at Eindhoven University of Technology. The former is the tool that contains digital filters as components using which a standard control block scheme can be formed, together with the original components. DCTOOLS© inspired a part of the S&C Library. The latter one is a dedicated, graphical loopshaping tool for motion systems and a snapshot of it is given in Figure 2.4. Despite the fact that they are free, they do depend on commercial environments, which limits their usability.

Furthermore, another widespread tool is LabVIEW©. It is a graphical programming environment used to develop complex measurement, test, and control systems using icons and connections between them, that resemble a flowchart. It offers integration with hardware devices and provides built-in libraries for advanced analysis and data visualization. It

---

3Copyright © 2002 René van de Molengraft
4Copyright © 2005-2009 DJH Bruijnen, G Witvoet
5Copyright © 2011 National Instruments Corporation
is similar to Simulink™, however it is more general and is primarily oriented toward hardware interfacing, whereas Simulink™ is mainly used for modeling and simulating. Lately, the latter changes and usage of Simulink™ for the control of the hardware increases.

On the other hand, some open source environments are present as well. Scilab™ is a good example. It is an open source, cross-platform numerical computational package, that can be used, similar to MATLAB™, for signal processing, statistical analysis, numerical optimization, modeling and simulation, etc. Toolboxes are also available as add-ons, from which modeling and control toolbox stands out. Nevertheless, the functionalities offered by it are limited and not generic, i.e., the tools are still in the process of the development. Moreover, the tools are mostly developed for control design and not real-time running, which is a very important aspect too.

Another interesting framework is OROCOS© (Open Robot Control Software), a general-purpose, free software, and modular framework. It is an environment for robot and machine real-time control and consists of a number of libraries, namely the Real-Time Toolkit, the Kinematics and Dynamics Library, the Bayesian Filtering Library and the Orocos Component Library. As an extremely efficient and convenient environment for robot control, it can be a good rolemodel for the S&C Library, following successful ideas and concepts for real-time implementation. Additionally, the application and utilization of the library through OROCOS© will be shown in the later phase of the report.

Complementary to OROCOS©, there is OROMACS. It is an acronym for Orocos-based Implementation Framework for Multi-Agent Controller Systems (MACS) and it is a project that uses OROCOS© components to develop MACS based controller software implementation framework for electromechanical motion systems. It combines the open architecture, run-time reconfigurable and structured controller design from MACS and the real-time, lock free data communication and thread-safe time determinism of OROCOS©, but tries to come up with a better performance controllers. The greatest advantage is that it is equipped with a console tool to navigate through OROCOS© applications, which is otherwise time consuming and single view. Also, it has a Graphical User Interface (GUI) software tool that can display the hierarchy of MACS in a tree view, browse or navigate through the hierarchy, accept and execute user commands, observe the contents of controller-agents involved in the controller application and plot the run-time data flow between the controller-agents.

Each of the aforementioned software products has its own advantages. Nonetheless, most of them are software environments and like that, they are already relatively specific, while the other tools, like for instance ShapeIt©, are environment-dependent (MATLAB™). The aim of this project is to try to emerge a level higher and develop such a general library that can be utilized by all of the mentioned frameworks, in the context of systems and control. Still, an important aspect of the library is compatibility with all those environments and therefore it is essential to be aware of them and to likewise design and develop the library. The idea is depicted in Figure 2.5. The requirements that would enable this are formulated in the following section.

---

6Scilab™ is a registered trademark of INRIA
7Copyright © 2002-2007 Herman Bruyninckx, Peter Soetens
2.3 Systems and Control Library Requirements

This part of the report elaborates S&C library requirements. In other words, the aspects of the library that need to be fulfilled are given, together with the Software Requirements Specification (SRS), which is a complete description of the wanted behavior of the library that is being developed.

2.3.1 Considered Aspects

During software design phase, a number of software aspects should be considered. However, first they need to be specified and that is done exactly at this point, i.e., in the process of requirements definition. These aspects are extremely important, since they reflect the goals the software is trying to achieve and determine how it will behave once it is developed.

The aspects that are considered in the design process of the S&C Library are the following:

- **Compatibility**
  The software is able to operate with other products and with various environments, which expand the potential library application. Also, a piece of software should be backward-compatible with its older version, enabling users to only update the library, without need to extensively adapt.

- **Expendability**
  New functionalities can be added to the library without major changes to the underlying architecture, which makes the further development of the library easy.

- **Modularity**
  The resulting software comprises well defined, independent components, which leads
to better maintainability. This also allows components to be implemented and tested in isolation before being integrated to form a desired software system.

- **Reliability**
  The software is able to perform a required function under stated conditions for a specified period of time. This aspect in a way determines the performance of the library.

- **Fault-tolerance**
  The library is resistant to component failures and is able to recover from them, anticipating the possible fault-states or inputs.

- **Reusability**
  The software is adaptable and flexible by means of parameters and different functions to the specific application. On top of that, the default values for the parameters have to be chosen in a manner most appropriate to its functionality, i.e., that would suit the majority of users.

- **Real-time operation**
  The library must be able to support real-time utilization, and hence algorithms must be carefully selected and feasible. Special attention should be addressed to avoidance of dynamic memory allocation, if possible, which hampers real-time usage.

### 2.3.2 Software Requirements Specification

Software Requirements Specification (SRS) describes the wanted behavior of the library. Possible library applications are used for the reference.

![Examples of possible Systems and Control Library applications](image)

**Figure 2.6:** Examples of possible Systems and Control Library applications

As the idea of the S&C Library was initiated for the purpose of robot control, robotics is naturally the first field of application. The library can be used for various general functionalities, like joint control, trajectory and path planning, obstacle avoidance, etc. Moreover, it can be used for specific tools as well, like watchdog units and solvers for kinematic models.
of mobile platforms (geometric decoupling), that take into account over- and under-actuated cases. This robotics application specifies an important requirement to the library, which is a real-time operation.

However, the intention is not only to keep the library limited to robotics, but also applicable in other fields. Motion control, like for example control of lithography systems, is one of them. In need to cope with aggressive motion profiles, it requires fast and reliable controllers, implemented on low-level like digital filters (PIs, lowpass filters, etc.). Since the intention of the S&C Library is to be applicable in this field too, those features are required in it. One would also prefer to be able to tune parameters of those. For that, certain control design tools should be present, like pole placement or loopshaping. Additionally, more requirements rise up, like decoupling of rigid and flexible body modes and MIMO (Multiple-Input Multiple-Output) control techniques. Here, a more general requirement should be addressed. The library should handle all types of models, namely transfer function models, both denominator and numerator representation and zero-pole representation, and state space models. The latter opens opportunities for numerous types of control techniques, like adaptive control, Kalman and Bayesian filters or robust control. Some examples of possible applications are presented in Figure 2.6.

![Figure 2.6](image)

**Figure 2.7:** Example of possible S&C Library applications: Optimization

Furthermore, feasibility of optimal control (see Figure 2.7), a control technique in which the control signal optimizes a certain "cost function", sets particular requirements as well. For instance, as part of it, model predictive control, one the most used techniques in process control, demands features for finding a minimum of the aforementioned "cost function". Moreover, system identification (see Figure 2.8), as one of most important parts of control engineering, should be possible as well. Various tools that are involved in identification, as noise generators, transfer function estimators, specific trajectory generators, etc. They should be present in S&C Library too.

**Summary**

To sum up, the requirements specification of a library depends strongly on its application. In the previous section, a brief overview of possibilities is presented, together with the
corresponding demands the applications introduce. Herein, a structured list of the foreseen tools is given. The first two items, namely digital filters and geometric decoupling of wheeled mobile platforms, are implemented within this project and will be elaborated in the following chapters. The rest is left for future work. However, the basis set in Chapter 2 of this report should enable newcomers to implement those with ease.

- Digital filters
- Geometric decoupling of wheeled mobile platforms for under- and over-actuated cases
- Transfer function and state space models
- Optimization tools
- MIMO control oriented tools
- Rigid and flexible modes decoupling
- System identification tools
- Watchdog units
- Loopshaping and pole placement tools, etc.

## 2.4 Software Design

Software design is the process of planning for a software solution. After the requirements and the specifications of software are determined, a plan for a solution needs to be designed. It includes low-level component implementation issues as well as the architectural view. An accent will be put on the software architecture. It represents a set of structures needed to reason about the system, which comprises software elements, relations among them and properties of both. Aiming at fulfilling the requirements set from the previous section, the so-called 5Cs separation of concern is chosen and it will be minutely explained in this
section. Moreover, in the second part of this section, the type of software library is chosen, based on the facts stated at the beginning of this chapter. Finally, the choice of programming language is analyzed, yielding the type of programming that will be used, i.e., component oriented programming. This section will be rounded up by a generic example of a Systems and Control Library component, which should sum up and concretize the software design decisions.

2.4.1 5Cs Approach

General

In order to make the design of large-scale software systems like the S&C Library possible, certain concerns need to be separated. The idea originates from [48], where the problems that had not been encountered in sequential programming were recognized. Here, the 4Cs approach was proposed, namely it suggested the concerns separation in Communication, Computation, Configuration and Coordination. Later, this concept was slightly augmented in [14]: the Configuration aspect is divided in Connection, which represents rules on how components are interconnected, and Configuration, which characterizes the knowledge regarding the best values to give to all configurable parameters in a component. Hence, the name 5Cs approach. The efficiency of this idea has been strongly established in the software design of complex systems, like in distributed systems platforms, such as CORBA (Common Object Request Broker Architecture) [38]. On top of it, the level of modularity and reusability of the software is extensively high. These facts motivated the choice of this approach, since it enables accomplishment of the determined requirements.

The 5 concerns are defined by:

1. Communication
   This layer handles the exchange of data and brings it to the Computational components that require it. It is the only part that is application independent.

2. Computation
   This is the crucial part of the system, as it executes all of its functionality. It provides an implementation of knowledge that is essentially the real added value of the overall system. All true computation ultimately takes place in elemental particles which have a built-in behavior. More complex particles of the code utilize these primitive ones by sending messages to them. This process enables complex behaviors and thus performing of complex computations. Also, this implies that the Computation depends on the Communication layer because of the exchange of that information.

3. Connection
   This concern specifies which Computation components should communicate with each other. Therefore, it strongly depend on Computation layer.

4. Configuration
   This functionality allows users of all the other concerns to influence their behavior and performance, by giving concrete values to the provided component-specific parameters. For instance, one can tune control or estimate gains, determine the Communication channels and their interaction policies, etc. For that reason, this layer depends on all other layers.
5. Coordination

This layer defines the concept on which all the components in the system function together. Coordination requires the Connection layer, since before specifying the concepts of interaction, the connections need to be specified.

![Diagram of 5Cs of separations of concerns]

The separation of concerns, as a layered structure, can be depicted as in Figure 2.9. The layer division is clear and easily understandable. The layer that is on top of the other one depends on it. The one below does not and should not need to know about higher layers. That leads us to do same conclusion as before, that Communication is application independent and that the Configuration depends on all of the other layers.

Directed Acyclic Graphs (DAGs)

Before the 5Cs separation of concerns concepts is explained in terms of the S&C Library, it is relevant to make a clear definition of Direct Acyclic Graph (DAG), as it is one of the essential segments of all the concerns. In computer science terminology, a DAG is, as the name suggests, a directed graph with no directed cycles [63]. In other words, it is formed by a collection of vertices and edges, each edge connecting one vertex to another, such that there is no way to start at some vertex and follow a sequence of edges that eventually loops back to the same vertex again. A simple example is given in Figure 2.10. One type of DAGs is polytree and it is a DAG for which there are no undirected cycles, i.e., it is formed by enhancing the DAG by giving a direction to each edge.

Systems and Control Library context

In the context of S&C Library, the same five concerns are separated. This can be explained through control block diagram structure, which consists of control blocks. Figure 2.11 depicts a standard feedback scheme.

1. Communication

Each individual control block is designed in a component based way, leading to component oriented programming, which will be elaborated in the following section. Each component will have an update function that reads the "input ports" (possibly including configuration of the ports as well), does the computations, and then writes the "output ports". This concern can be stand-alone and is implemented at the
moment as such, but can also be enhanced, such that it depend on some existing well-functioning tool. It allows usage in the fully collocated deployment, i.e., deployment where everything takes place in one single thread, without any middleware communication. This means that each Connection is just a share variable, each Port access is read/write to that connection and Coordination of the computational scheduling is executed via the Directed Acyclic Graph (DAG) data structure. This is the preferred way, however DAG structure will not be designed in this project, since it is much more general and other available tools can be utilized for this purpose. Moreover, it can also be augmented for some other case like multi-threaded deployment, multi-process deployment or multi-platform deployment.

2. Computation
Control block diagrams have one generic behavior, i.e., the whole diagram is computed in every sampling instant. However, one has to come up with an explicit partial ordering in which the block would be called in a structured way, since difference in execution (calling the update function of each block) order could cause different results. A DAG, or more specifically a polytree (see [49]) is an appropriate data structure to encode such a partial ordering, because it, by definition, requires sequential execution of the blocks. This also relates to the Communication concern, explained before.

3. Connection
Control block diagrams are graphs in general and like those, require a data structure that represents this. Control diagrams typically also have a hierarchical decomposition in that one control block can consists of a complete control block diagram in itself (subsystem, see Figure 2.13). Hence, the concept of composite control block must be introduced as a basic class in the library.

4. Configuration
Each individual control block component must be appropriately initialized and finalized, especially when the control scheme is used in a real-time environment, where the actuators are control. Apart from these functions in single components, the same should be present in composite ones as well. The Coordination state machine (see Coordination part) of the whole control diagram should execute this configuration,
i.e., each initialize, finalize and update function of composite control block should take care of the corresponding initialization, finalization and updating of the control diagram that it encapsulates. When using DAG as a data structure, this yields to be rather straightforward due the fact that the data structures can be nested hierarchically. Additionally, run-time configuration of the properties of the blocks that constitute the composition should be possible. This should be done in the update function of the composite control block, but via the configuration interfaces of the individual control blocks.

5. Coordination

No pure state machine functionality should be present in the S&C Library, since that is the responsibility of the environment from which the functionality will be utilized. Therefore, a Coordination state machine should be used (again a DAG data structure).

Remark should be made here about DAG (or polytree) data structure, in the context of S&C Library. It is used as a collection of tasks that must be ordered into a sequence, since the result of performing certain tasks will influence the behavior of the others. Particularly, each vertex of a DAG stands for one task and each edge for a constraint that determines the order of execution. To sum up, the DAG will enable S&C Library to have fully collocated deployment in terms of Communication concern, component partial ordering in terms of Computation concern and composition of components in terms of Coordination.

2.4.2 Selection of Library Type

As stated before, a software library can be defined as a collection of resources used to develop software [56]. Among several types, the two that point up as most used are static and shared libraries. This section will summarize the pros and cons of both and give conclusion which one is more suitable in case of the Systems and Control Library.

The crucial difference of the two is the following: a static library consists of a set of routines which are copied into a target application by the compiler or linker, producing object files and a stand-alone executable file, while a shared library involves loading the library subroutines into an application program at load-time or run-time, rather than linking them in at compile time. Additionally, in case of static libraries loading of all code into actual run-time memory locations is common, which can take even more time than the compilation process and must be performed each time when any of the modules is recompiled. On the other hand, shared library only records library routines which the program needs and the index names or numbers of the routines in the library. At the appropriate time the loader adds the relevant data to the process’s memory space. This enables applications that utilize the library to automatically take advantage of improvement in the library, due to their dynamic
2.4. Software Design

linkage. In other words, the functionality of the client applications can be improved and extended without requiring application developers to recompile the applications.

Having in mind everything stated above, shared library imposes as a much more flexible and sophisticated solution and therefore is suggested in case of the S&C Library.

2.4.3 Programming Language Choice

In the software design phase, the question regarding the programming language that is going to be used in S&C Library emerges. A programming language is an artificial language designed to express computations that can be performed by a computer [4]. Thousands of different languages nowadays exit, however only a few are the real candidates for the purpose of this library. For instance, low-level languages like assembly or VHDL language provide (almost) no abstraction with respect to machine language and they strongly depend on computer (or microprocessor) architecture [13]. On the other hand, languages like Java™ with strong abstraction are more user-friendly, but due to that high level of abstraction they cannot guarantee execution speed, which is usually rather slow, and they make memory consumption too high, i.e., nonoptimal, which greatly hampers real-time usage [61]. The choice is therefore naturally narrowed to two, namely C and C++ programming languages. This was concluded by a petite literature survey and the final decision was motivated by the work of Bruyninckx [13].

The C language [31], as the one that most operating systems are programmed in (UNIX before all others), is generally a standard. It is designed to provide (relatively) low-level access to memory, but also to encourage cross-platform programming. Its foremost advantage is the pointer concept, where a pointer is a programming language data type whose value refers directly to, i.e., "points to" another value stored elsewhere in the computer memory using its address. This enabled placing a variable of a program onto a specific address in memory. Moreover, it is notably efficiency, due to no run-time in which non-deterministic operations happen outside the control of the programmer.

On the other hand, there are also couple of drawbacks of C language. The most important one is that it does not allow Object-Oriented Programming (OOP). OOP is a programming paradigm using "classes", i.e., data structures consisting of data fields and methods together with their interactions. Another important concept is an "object", which is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable. This type of programming is introduced in C++ language, among others. C++, however, rises above the others, due to its correspondence with C language, keeping the majority of its advantages, namely pointers and efficiency [32].

The following three primary aspects of object-oriented programming make C++ programming language a rational choice:

- Encapsulation of the implementation
  Encapsulation enables a structure with protection and authorization. It contributes to distinguishing between interface and implementation, where the former represents the specification of the set of operations that can be performed on the object and the latter has a data part and a procedural part. The data part is the representation or state of the object and the procedure part describes the implementation of each operation.

---

8 Java™is a registered trademark of Oracle and/or its affiliates
Moreover, encapsulation enables modularity, which is crucial for complex software design.

- **Inheritance**
  Inheritance allows to create classes which are derived from other classes, so that one class becomes the parent of another class. Also, this means that a “child” class automatically includes some of its “parent’s” members, plus its own. Additionally, multiple inheritance is possible, i.e., it is possible that a class inherits members from more than one class. For instance, a class `Polygon` is a parent class of classes `Rectangle` and `Triangle`, since they share some of the same features (area, height, etc.). However, they also have some different characteristics, like formula for calculation of the area or perimeter.

- **Polymorphism**
  This is the concept which allows the same software objects to behave in diverse manners, depending on various parameters or context where it is utilized. For example, the plus (+) operator can be used for addition of two integer variables (3+5), but also for addition of two double variables (3.14+2.0).

### 2.4.4 Component Oriented Programming

Component oriented programming (COP) enables software to be assembled from prebuilt software components, which are reusable, segmented blocks. These components have to follow certain predefined standards including interface, connections, versioning, and deployment [66]. Components come in all shapes and sizes, ranging from small application components to the so-called large grain components that contain extensive functionality. In principle, every component should be reusable independent of context. COP is similar to object oriented programming (OOP), but it expands the paradigm a bit [23]. While OOP only emphasizes on classes and objects, COP emphasizes also interfaces and beyond all, composition. This means that the clients in COP do not need any knowledge of how a component implements its interfaces. Therefore, interfaces should remain unchanged, so the clients are not affected by changes in interface implementations. Moreover, COP provides an effective way to follow the software engineering principle of dealing with change. Components are easy to adapt to new and changing requirements. Moreover, COP is compatible with the 5Cs concept and therefore fully meets the requirements of the S&C Library.

Software components are defined in various ways from different points of view. Several definitions of a software component are, see [12]:

1. "A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture. A component conforms to and provides the physical realization of a set of interfaces. (Philippe Krutchen, Rational Software)"

2. "A run-time software component is a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at run-time. (Gartner Group)"

3. "A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to third-party composition. (Clemens Szyperski)"
All of these definitions together compose a rather easy understanding of a term component.

2.5 A Systems and Control Library Component

In this section a clear representation of a S&C Library component is given. All of the software design decisions made in the previous section should come together here. It should also enable future developers to understand those decisions better, through concrete implementation. An example is depicted of a component in Figure 2.12.

Each component has four main public functions, also called “hooks”, namely initialize, update, finalize and configure. They have the following functionality:

- Initialize
  This "hook" initializes each component, sets default parameters and initial values to the variables it consists of. It is necessary to run it at the beginning of a component deployment. It can also be used in case of a component fault, to bring it into a consistent state.

- Update
  This is the part where all the Computation takes place. Computation is preceded by reading of the input variables and followed by calculating (updating) the output ones. The Computation directly depends on the parameters, set through the configure function.

- Finalize
  This function finalizes each component, i.e., it saves the needed data and sets the default values to the output variables. This is crucial during real-time control, due to the online actuation of motors, which could be damaged in case the finalization is not performed in the proper manner.
Chapter 2. Software Development Process

- Configure
  The "hook" that takes care of parameters of a component, which determine its behavior. It contributes to the reusability and adaptability of the library. In the case of composition, this "hook" should make sure all the single components are configured properly and that they are run (updated) in required sequence.

These four functions are mandatory part of each S&C Library component. They reflect the 5Cs approach and enable fulfillment of the set software requirements, like reusability, expendability, adaptability, etc.

Apart from those main functions, there are some auxiliary public functions as well. Mostly, they are pure get functions (getters), and in some special cases set functions (setters). The former enable users to access the values which are results of Computation, like for instance component output value. The latter are not as common, but if there is need, they can be present in the class definition. One example is the function that sets the value of epsilon, which determines the precision of a numeric data type (e.g., used for comparison between variable of type double and zero).

Private members of classes are usually variables and functions that are used for Computation. They cannot be seen or accessed by users, and they usually implement the interface represented by the public functions.

All these public and private members round up a component. An example of it, with all the needed explanations, is given in Appendix D.

![Figure 2.13: Component oriented programming block diagram extended](image)

As stated before, compositions of components should be possible as well. This is done using DAG or polytree and it is presented in Figure 2.13. The components are linked through Connections, which have the predefined Communication feature. They are also read/write, so the overview of the composition can be made. Furthermore, this enables visualization of the components that are running, their connection and current configuration. Yet, the
composition is nothing more than another component, that has input and output port(s) and can be further linked to other components.

2.6 Licensing and Copyright

2.6.1 Licensing

The development of the Systems and Control Library is an open-source project. The goal of the library is to support a wide variety of users and developers, from academia and research to industry and commercial use of the code. Hence, a license that enables that needs to be chosen.

The preferred license for the project is the BSD License [15], in particular FreeBSD License. Licensing all the new code under BSD is strongly encouraged. All currently written code is licensed under BSD.

There are many factors that influenced the choice of BSD.

- BSD allows any kind of reuse, from academic to commercial, open or closed.
- BSD is compatible with all other OSI-approved (Open Source Initiative) licenses, unlike some other (e.g., GNU General Public License - GPL).
- BSD does not require that changes be contributed back. However, there are large thriving communities using the BSD license and members of these communities frequently contribute improvements, without being required to do so.
- If needed, licensing can always be upgraded to some other, more restrictive OSI-approved license.

Every source file should contain a commented license summary at the top. An example is given in Appendix D as a part of the presented header file.

2.6.2 Copyright

Under the Berne Convention [2], the author of a work automatically holds copyright, with or without a formal statement to that effect. However, making copyright explicit is helpful in long-term project management.

Each source file should contain a commented copyright line at the top, e.g., "Copyright (c) <year> <copyright holder>". This statement is usually directly above the license summary. In the case of Systems and Control Library, copyrights are held by Eindhoven University of Technology (TU/e).

Again, an example is given in Appendix D as a part of the presented header file.
2.7 Documentation

Software documentation or source code documentation is written text that accompanies computer software. It explains how it operates and how to use it and it is intended for both developers and users.

The chosen documentation generator for this project is Doxygen [21]. Doxygen is a tool for writing software reference documentation for the programming languages C and C++, among others. It is a cross-platform tool, i.e., it runs on most Unix-like systems, including Mac OS X, and on Windows. The most appealing thing regarding Doxygen is the fact that documentation is written within code, and is thus relatively easy to keep up to date. It can also cross reference documentation and code, so that the reader of a document can easily refer to the actual code. It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual from a set of documented source files. There is support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF and compressed HTML pages as well. Furthermore, one can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.

Doxygen is free software, released under the terms of the GNU General Public License.
Chapter 3

Digital Filters

Digital filters are the most commonly used and implemented tools in the field of control. Thus, they also represent the foundation of S&C Library and they were chosen to be the first segment of it. In this chapter, a theoretical background is given, starting with the definition of a filter, its relevant properties and stability issues. This is followed by the elaboration of possible filter realization methods and discretization methods. All common methods are analyzed, together with their advantages and disadvantages. Finally, various filters derivations are presented. In this chapter, for the sake of report flow, only final results are given, while the full derivations can be found in appendices.

3.1 Filter Definition

In this section, an introduction to digital filters is given, starting with the definition of a digital filter, which is followed by the declaration of all relevant properties and stability conditions.

In electronics, computer science and mathematics, a digital filter is a system that performs mathematical operations on a sampled (discrete-time), quantized signal to reduce or enhance certain aspects of that signal. A digital filter is characterized by its transfer function, or equivalently, its difference equation.

The difference equation is a formula for computing an output sample at time $n$ based on past and present input samples and past output samples in the time domain. The general, causal, linear, time-invariant (LTI) difference equation is represented as follows:

$$y(n) = b_0x(n) + b_1x(n - 1) + b_2x(n - 2) + \cdots + b_Nx(n - N)$$
$$- a_1y(n - 1) - a_2y(n - 2) + \cdots + a_My(n - M),$$

$$= \sum_{i=0}^{N} b_i x(n - i) - \sum_{j=1}^{M} a_j y(n - j). \quad (3.1.1)$$

where $x$ is the input signal, $y$ is the output signal, and the constants $b_i, i = 0, 1, 2, \cdots, N$, $a_j, j = 1, 2, \cdots, M$ are called the coefficients. When the coefficients are real numbers the
filter is said to be real. Otherwise, it may be complex. However, in this analysis only the former is taken into account.

Notice that a filter of the form of Equation 3.1.1 can use past output samples (such as \( y(n-1) \)) in the calculation of the present output \( y(n) \). This use of past output samples is called feedback. On the other side, using past and present inputs scaled by some coefficients is called feedforward. More specifically, the \( b_i \) coefficients are called the feedforward coefficients and the \( a_j \) coefficients are called the feedback coefficients.

A filter is said to be recursive if and only if \( a_j \neq 0 \) for some \( j > 0 \). Recursive filters are also called Infinite-Impulse-Response (IIR) filters, since they have a non-zero impulse response function over an infinite length of time. When there is no feedback \( (a_j = 0, \forall j > 0) \), the filter is said to be nonrecursive or Finite-Impulse-Response (FIR) digital filter.

Furthermore, using mathematical analysis of the transfer function one can describe how the filter would respond to any input, i.e., describe transfer characteristics of a filter – hence the name. As such, designing a filter consists of developing specifications appropriate to the problem (for example, a first-order lowpass filter with a specific cut-off frequency), and then producing a transfer function which meets the specifications. For example, in control a continuous-time transfer function is initially generated, since the continuous-time controller design is more developed. Then, using discretization methods, which will be discussed further, a discrete-time transfer function is derived.

The discrete-time transfer function for a causal, LTI digital filter can be expressed as a transfer function in the \( z \)-domain, i.e., it is equal to the \( z \)-transform of the impulse response \( h(n) \). Discrete-time transfer function of a digital filter is represented in the following form:

\[
G(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}},
\]

(3.1.2)

where \( a_i \) and \( b_i \) are formerly mentioned feedback and feedforward coefficients, while \( M \) and \( N \) are orders of denominator and numerator, respectively. The \( z \)-transform converts a discrete, time domain signal into a complex frequency-domain representation. The order of an LTI filter equals the order of its transfer function, which is defined as the maximum of its numerator and denominator polynomial orders, namely \( \max\{M, N\} \). If the order of numerator is greater than the order of denominator \( (M > N) \), transfer function is called non-proper. Otherwise, it is called strictly proper (proper in case \( M = N \)). Digital filter can be realized only if its order is finite.

An alternative way to represent a digital filter is by its poles and zeros. This form can be obtained by factorization of denominator and numerator of Equation 3.1.2, which yields:

\[
G(z) = \frac{B(z)}{A(z)} = k \frac{(1 - z_{z_1} z^{-1})(1 - z_{z_2} z^{-1}) \cdots (1 - z_{z_N} z^{-1})}{(1 - z_{p_1} z^{-1})(1 - z_{p_2} z^{-1}) \cdots (1 - z_{p_M} z^{-1})},
\]

(3.1.3)

where \( z_{z_i} \) and \( z_{p_i} \) are zeros and poles of the filter and \( k \) is the gain of the filter. The filter order now equals the number of poles or zeros, whichever is greater. Additionally, a transfer function with no pole-zero cancelations is said to be irreducible, i.e., there are no equal zeros and poles.

At this point, it is also important to indicate conditions for filter stability. It holds that an irreducible transfer function \( G(z) \) and thus a digital filter that it represents, is stable if and only if its poles have magnitude less than one, i.e., each pole lies strictly inside the unit circle.
of the \( z \)-plane, \(|z_p| < 1\). In terms of an impulse response, this means that filter's impulse response \( h(n) \) decays to 0 as \( n \) goes to infinity. In contrary, if there is a pole outside of the unit circle of the \( z \)-plane, an exponentially increasing component of the impulse response would be present, which would lead the digital filter response to instability.

### 3.2 Filter Realization

Digital filters, introduced in the previous section, as digital systems require specific implementation or realization structure. This section introduces the four direct-form filter realization approaches and discusses the realization of filters as parallel or series combinations of smaller filter sections.

A careful study of filter forms can be important when numerical issues arise, such as when implementing a digital filter in a fixed-point processor. For implementations in floating-point arithmetic, for instance at 32-bit word-lengths or greater, the choice of filter realization structure is not critical, but it does usually influence the outcome.

On the other hand, for example, in MATLAB™ one rarely uses anything but the filter function, which is implemented in double-precision floating point (typically 64 bits and more bits internally in the floating-point unit) [58]. Eventually, the same realization structure is going to be implemented in this case, only before that, the review of all the possibilities is given.

#### 3.2.1 The Four Direct Forms

Difference equation (3.1.1) specifies the Direct-Form I (DF-I) realization of a digital filter, one of four direct-form structures to choose from. The Direct-Form II structure, another common choice, has a slightly different difference equation, see Equation (3.2.1). The other two direct forms are obtained by transposing Direct-Forms I and II [58]. This chapter elaborates all four of them.

**Direct-Form I**

As mentioned before, difference equation (3.1.1) specifies the Direct-Form I (DF-I) realization of a digital filter. The DF-I signal flow graph for the second-order case is shown in Figure 3.1.

The DF-I structure has the following properties:

1. It can be regarded as an \( N \)-zero filter section followed in series by an \( M \)-pole filter section.

2. In most fixed-point arithmetic schemes there is no possibility of internal filter overflow. That is, since there is fundamentally only one summation point in the filter, and since fixed-point overflow naturally "wraps around" from the largest positive to the largest negative number and vice versa, then as long as the final result \( y(n) \) is "in range", overflow is avoided, even when there is overflow of intermediate results in the sum. This is an important, valuable, and unusual property of the DF-I filter structure.
3. There are up to twice as many delays as are necessary (in oppose to Direct-Form II, see the following subsection). As a result, the DF-I structure is not optimal with respect to delay. In general, it is always possible to implement an \( N \)-th order filter using only \( N \) delay elements.

4. As is the case with all direct-form filter structures (those which have coefficients given by the transfer-function coefficients), the filter poles and zeros can be very sensitive to round-off errors in the filter coefficients. This is usually not a problem for simple (low order sections), but it can become a problem for higher order direct-form filters. This is the same numerical sensitivity that polynomial roots have with respect to polynomial-coefficient round-off. As it is well known, the sensitivity tends to be larger when the roots are clustered closely together, as opposed to being well spread out in the complex plane. This sensitivity can be minimized by factorization of filter transfer function into series and/or parallel of second-order sections.

A very useful property of the Direct-Form I realization is that as long as the output signal is in range, the filter will be free of numerical overflow. Most IIR filter realizations do not have this property. However, even though DF-I is immune to internal overflow, the noncanonical structure with respect to delay makes it not preferable for realization.

**Direct-Form II**

Another digital filter realization method is Direct-Form II (DF-II). The signal flow graph for it is shown in Figure 3.2.

The difference equation is different than the one used in the DF-I structure and can be written as:

\[
\begin{align*}
    v(n) &= x(n) - a_1 v(n - 1) - \cdots - a_M v(n - M), \\
    y(n) &= b_0 v(n) + b_1 v(n - 1) + \cdots + b_N v(n - N),
\end{align*}
\]  

(3.2.1)

which can be interpreted as a \( M \)-pole filter followed in series by a \( N \)-zero filter. This contrasts with the DF-I structure of the previous section in which the \( N \)-zero FIR section precedes the \( M \)-pole recursive section in series. Since LTI filters in series commute, this ordering may be reversed and implemented as an all-pole filter followed by a FIR filter in
3.2. Filter Realization

The signal flow graph for the Direct-Form II realization

Figure 3.2: The signal flow graph for the Direct-Form II realization

series. In other words, the poles may come first, followed by the zeros, without changing the transfer function. When this is done, it is easy to see that the delay elements in the two filter sections contain the same numbers. As a result, a single delay line can be shared between the all-pole and all-zero (FIR) sections.

In summary, the DF-II structure has the following properties:

1. It can be regarded as a $M$-pole filter section followed by a $N$-zero filter section.
2. It is optimal with respect to delay. This happens because delay elements associated with the pole and zero sections are shared and none can be omitted.
3. In fixed-point arithmetic, overflow can occur at the delay-line input, unlike in the DF-I realization. This is caused by the multiple summation points, which can all simultaneously reach the boundary value, and by that cause overflow at one of the lines.
4. As with all direct-form filter structures, the poles and zeros are sensitive to round-off errors in the coefficients $b_i$ and $a_j$, especially for high transfer-function orders. Lower sensitivity is obtained using series low-order sections (e.g., second order), or by using ladder or lattice filter structures [57].

Transposed Direct-Forms

The remaining two direct forms are obtained by formally transposing Direct-forms I and II [43]. This procedure is also known as flow graph reversal.

An important fact regarding the transposition is that it does not modify a transfer function of Single-Input, Single-Output (SISO) filter. The proof can be derived from Mason’s gain formula for signal flow graphs [36]. Moreover, it is particularly easy to find transposition of a filter. Only the direction of all signal paths need to be reversed, as well as signal branches and summers. Additionally, after transposition, the input signal will be on the right and the output on the left, unlike usual structure. In some cases, the whole diagram is mirrored, in order to get the natural flow chart.
Figures 3.3 and 3.4 show the Transposed-Direct-Form I (TDF-I) and Transposed-Direct-Form II (TDF-II) structure for the general IIR digital filter. For the ease of comparison with the normal direct forms, charts are left unmirrored.

![Figure 3.3: The signal flow graph for the Transposed-Direct-Form I realization](image)

![Figure 3.4: The signal flow graph for the Transposed-Direct-Form II realization](image)

**Numerical Robustness of TDF-II**

An advantage of the transposed Direct-form II structure (depicted in Figure 3.4) is that the zeros effectively precede the poles in the series order. As mentioned above, in many digital filters design, the poles by themselves give a large gain at some frequencies, and the zeros often provide compensating attenuation. This is especially true for filters with sharp transitions in their frequency response, such as the elliptic-function-filters, at which the sharp transitions are achieved using near pole-zero cancelations close to the unit circle in the z-plane. This makes TDF-II a good choice for the realization of digital filters needed for control.
3.2.2 High Order Filter Decomposition

Another huge improvement in terms of numerical performance can be made when it comes to high order digital filters. Namely, it is better to break down a digital filter \( H(z) \) into more first- and/or second-order elementary sections, either in series or in parallel [58]. In control theory, this is exactly what is being done – controllers are designed as a collection of low order filters (lead lags, lowpass filters, etc.). The two different approaches are laid out in this section.

Series Second-Order Sections

For many filter types, such as lowpass, highpass and bandpass filters, a good choice of realization structure is often series second-order sections. In fixed-point applications, the ordering of the sections can be important.

Parallel First and/or Second-Order Sections

Instead of decomposing a filter into a series of second-order sections, one can break the filter up into a parallel sum of first and/or second-order sections. Parallel sections are based directly on the partial fraction expansion (PFE) of the filter transfer function. Additionally, there is a FIR part when the order of the transfer-function denominator does not exceed that of the numerator, i.e., when the transfer function is not strictly proper. The most general case of a PFE, valid for any finite-order transfer function, is given by

\[
H(z) = F(z) + z^{-(K+1)} \sum_{i=1}^{N_p} \sum_{k=1}^{m_k} \frac{r_{i,k}}{(1 - p_i z^{-1})^k},
\]

where \( N_p \) denotes the number of distinct poles, and \( m_k \geq 1 \) denotes the multiplicity of the \( i \)-th pole. The polynomial \( F(z) \) is the transfer function of the FIR part.

Summary

In summary, the following general guidelines regarding series against parallel elementary-section realizations should be noted:

- Series sections are preferred when all sections contribute to the same passband, such as in a lowpass, highpass, bandpass, or bandstop filter.

- Parallel sections are usually preferred, on the other side, when the sections have disjoint passbands.

In this project, only elementary digital filters of first and second order are implemented. The composition of them is possible, while the type of it can be chosen depending on the application.
3.3 Filter Discretization

As it has been mentioned in the introduction of this chapter, discretization is needed in order to implement the digital filters for the purpose of control. This section motivates this and analysis possible discretization methods.

The design of a control system can be divided into two steps. First, the process or plant needs to be identified, yielding mathematical model form of the system or data that represents it, so that its behavior can be analyzed. That is followed by the design of an appropriate controller in order to get the desired response of the controlled system. In the continuous-time domain the controller is represented by differential equations, on which we elaborated before. Replacing a continuous controller into discrete-time form is always an approximation of the continuous system. A schematic overview of this process is given in Figure 3.5.

The different discretization methods yield different digital controller performance.

An alternative to this approach is to gather discrete-time information of the system and directly compute a digital controller. However, despite the fact that this common in optimal or robust control techniques, it is extremely difficult for manual loopshaping.

3.3.1 Survey of Discretization Methods

Modern controllers are realized by digital circuits based on microprocessors or process-control computers. These devices can be characterized by discrete operations, where the control algorithms are implemented by a computer program. The most important parameter of these systems is the sampling time. Thus, these systems are called discrete-time systems. Moreover, due to the fact that the processors have finite word length, the signals are discretized in amplitude, but also encoded. Two different approaches can be used for controller design. The first strategy is based on the continuous-time system model or measured data, where the controller design is carried out also in the continuous-time domain, see Figure 3.5. The second method uses a discrete system model and the design procedure is executed in the discrete-time domain. As stated before, the first case allows a number of manual loopshaping design techniques to be used. Hence, discretization methods play a huge role in the controller design process. Moreover, since they yield different results, stressing their differences and analyzing them thoroughly is very important.

The methods presented here are Euler backward and forward differentiation method, step invariant or zero order hold (ZOH) method, bilinear transformation and matched pole-zero method.

If this is put in terms of a transfer function, the idea is to transform continuous-time transfer function \( G(s) = \frac{B(s)}{A(s)} \) to discrete-time one \( G(z) = \frac{B(z)}{A(z)} \).

\[
G(s) \rightarrow G(z),
\]  

(3.3.1)
where arrow represents transformation from $s$-domain to $z$-domain as shown in Equation (3.3.2), i.e., each $s$ in $G(s)$ is substituted by a function of $z$, in order to get $G(z)$.

$$s \approx f(z).$$  
(3.3.2)

Couple of remarks should be made here. Firstly, the expression $z = e^{sT_s}$ is used, which is a $z$-transform, and as that a generalization of the discrete-time Fourier transform (DFT) \[44\]. Secondly, the assumption that initial condition is zero is made ($x(0) = 0$) for all the methods.

**Euler Backward and Forward Differentiation Methods**

The first method is commonly used in discretizing simple signal filters and industrial controllers, while the second one is used in developing simple simulators \[28\]. Both methods are time domain approximations. The forward differentiation method is somewhat less accurate than the backward differentiation one, but it is simpler to use. Particularly, with nonlinear models the backward differentiation method may give problems since it results in an implicit equation for the output variable, while the forward differentiation method always gives an explicit equation. However, the forward method for stable continuous systems does not always result in stable digital equivalents, while the backward one does. The backward method has though another drawback – it sometimes transforms instable behavior to a stable one, which can be unwanted. Moreover, higher sampling frequency naturally gives a better approximation.

The forward differentiation method can be seen as the following approximation of the time derivative of a time-valued function which here is denoted $x$:

$$\dot{x}(t_k) \approx \frac{x(t_{k+1}) - x(t_k)}{T_s},$$  
(3.3.3)

where $T_s$ is the time step, i.e., the time interval between two subsequent points of time. The name "Forward differentiation method" stems from the $x(t_{k+1})$ term in Equation (3.3.3).

Now, the Laplace transform is performed on Equation (3.3.3), which leads to transformation $s \approx f(z)$:

$$sX(s) - x(0) \approx \frac{e^{sT_s} - 1}{T_s} X(s).$$  
(3.3.4)

Since $x(0) = 0$ and $z = e^{sT_s}$, the following holds:

$$s \approx \frac{z - 1}{T_s} \iff z \approx 1 + sT_s.$$  
(3.3.5)

The forward method maps the whole left half $s$-plain into the plain left of $x = 1$ in the $z$-plain, see Figure 3.7. Thus, it preserves stability.

On the other hand, the backward differentiation method is based on the following approximation of the time derivative:

$$\dot{x}(t_k) \approx \frac{x(t_k) - x(t_k - 1)}{T_s}.$$  
(3.3.6)
The name "Backward differentiation method" stems from the $x(t_k - 1)$ term in Equation (3.3.6).

Now, the Laplace transform is performed on Equation (3.3.6), which leads to transformation $s \approx f(z)$:

$$sX(s) - x(0) \approx \frac{1 - e^{-sT_s}}{T_s} X(s).$$

(3.3.7)

Since $x(0) = 0$ and $z = e^{sT_s}$, the following holds:

$$s \approx \frac{1 - z^{-1}}{T_s} \approx \frac{z - 1}{zT_s} \iff z \approx \frac{1}{1 - sT_s}.$$

(3.3.8)

It maps the whole left half $s$-plain into the interior of the circle with radius 0.5 and center at $(0.5, 0)$ in the $z$-plain, see Figure 3.8. Thus, it preserves stability, but since the poles
are confined to a relatively small set of frequencies, no highpass filter is really possible to implement.

![Diagram](image.png)

**Figure 3.8:** Mapping from \( s \) to \( z \) plane of the backward differentiation method

### 3.3. Filter Discretization

**Zero Order Hold (ZOH) or Step Invariant Method**

Zero order hold (ZOH) is a time domain approximation and it means that the physical input signal to the system is held fixed between the discrete points of time. The other name (step invariant method) comes from the fact that it tries to obtain a discrete-time filter \( G(z) \) which is equivalent to the given continuous filter with transfer function \( G(s) \) such that the filters have the same step response at the sampling instants. Hence, the following is required:

\[
Z^{-1}[G(z)(1 - z^{-1})] = L^{-1}[G(s)(\frac{1}{s})]|_{t=kT_s},
\]

(3.3.9)

where \( Z^{-1} \) and \( L^{-1} \) are inverse \( z \)-transform and inverse Laplace transform, respectively.

Now taking the \( z \)-transform of both sides yields

\[
G(z)(1 - z^{-1}) = Z\{L^{-1}[G(s)(\frac{1}{s})]|_{t=kT_s}\}.
\]

(3.3.10)

Next, it should be noted that when it is stated that a \( z \)-transform of a continuous signal is taken, actually a \( z \)-transform of the discrete-time signal obtained by evaluating the continuous signal at the sampling instants is taken instead. Accordingly, the evaluation of the inverse Laplace transform of \( G(s)/s \) at \( kT_s \) is implicit in the definition of the \( z \)-transform, so it does not really need to be specified in Equation (3.3.10) above.

\[
G(z)(1 - z^{-1}) = Z\{L^{-1}[G(s)(\frac{1}{s})]\} = Z\{G(s)(\frac{1}{s})\}.
\]

(3.3.11)

By rewriting the Equation (3.3.11) so that \( G(z) \) is extracted, the following is obtained:

\[
G(z) = (1 - z^{-1})Z\{G(s)(\frac{1}{s})\};
\]

\[
G(z) = (\frac{z - 1}{z})Z\{G(s)(\frac{1}{s})\}.
\]

(3.3.12)

This discretization method is relatively complicated to apply. Moreover, delay caused by the ZOH method is its main disadvantage. Essentially, it introduces phase lag and distorts the frequency response of the controller. Therefore, it is generally not used for filters. Still, it is implemented in the library and a potential usage is left to users.
Chapter 3. Digital Filters

Bilinear Transformation

The bilinear transform is used in discrete-time control theory to transform continuous-time system representations to discrete-time and vice versa. It is a time domain approximation and can be seen as a correction of backward differentiation method. The term bilinear is related to the fact that the imaginary axis in the complex $s$-plane for continuous-time systems is mapped or transformed into the unity circle for the corresponding discrete-time system \[27\]. Another valuable property of the bilinear transform is that order is preserved, i.e., an $N$th-order $s$-plane transfer function carries over to an $N$th-order $z$-plane transfer function.

This method is also known as trapezoid approximation method or Tustin’s method. The reason for that is that it is based on the trapezoidal approximation of the time derivative:

\[
\frac{\hat{x}(t_k) + \hat{x}(t_{k-1})}{2} \approx \frac{x(t_k) - x(t_{k-1})}{T_s}. \tag{3.3.13}
\]

Now, the Laplace transform is performed on Equation (3.3.13), which leads to transformation $s \approx f(z)$:

\[
\frac{1 + e^{-sT_s}}{2} sX(s) - x(0) \approx \frac{1 - e^{-sT_s}}{T_s} X(s). \tag{3.3.14}
\]

Since $x(0) = 0$ and $z = e^{sT_s}$, the following holds:

\[
s \approx \frac{2}{T_s} \frac{1 - z^{-1}}{1 + z^{-1}} \iff z \approx \frac{1 + sT_s/2}{1 - sT_s/2}. \tag{3.3.15}
\]

It maps the whole left half $s$-plain into the unit circle in the $z$-plain and thus preserves stability, see Figure 3.9.

\[\text{Figure 3.9: Mapping from } s \text{ to } z \text{ plain of the bilinear transformation}\]

Despite the resemblance of stability under this method, the scheme suffers from a severe drawback: the entire imaginary axis is mapped to only $2\pi$-length of the unit circle. This is a great amount of distortion, which is overcome by introduction of an extended Tustin’s rule. In general the frequency responses of $G(s)$ and of $G(z)$ are not equal at the same frequencies. Tustin’s method can be modified or enhanced so that one can obtain equal frequency response of them both at one or more predefined critical frequencies \[22\]. This procedure is called prewarping and the modified method is called Tustin’s method with prewarping. Essentially, this method is a combination of time and frequency domain approximation.

Tustin’s method with prewarping:

\[
s \approx \frac{1 - z^{-1}}{\alpha} \frac{1 + z^{-1}}{1 + \frac{s}{\alpha}} \iff z \approx \frac{1 + sT_s/\alpha}{1 - sT_s/\alpha}. \tag{3.3.16}
\]
3.3. Filter Discretization

where \( \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)} \) and \( \omega_p \) is prespecified frequency at which the two transfer function, namely continuous and discrete one, match.

This becomes extremely useful when it comes to discretization of filters, since most of them have only one frequency at which the match is crucial.

Example:
Suppose a sinusoid at the continuous frequency of \( s = j\omega_p \), which corresponds to the discrete frequency of \( z = e^{j\omega_p T_s} \), is applied to the systems. We have:

\[
G_z(e^{j\omega_p T_s}) = G_s\left(\frac{\omega_p}{\tan(\omega_p T_s/2)} \frac{e^{j\omega_p T_s} - 1}{e^{j\omega_p T_s} + 1}\right)
\]

\[
= G_s\left(\frac{\omega_p}{\tan(\omega_p T_s/2)} \frac{e^{j\omega_p T_s/2} - e^{-j\omega_p T_s/2}}{e^{j\omega_p T_s/2} + e^{-j\omega_p T_s/2}}\right)
\]

\[
= G_s\left(\frac{\omega_p}{\tan(\omega_p T_s/2)} \sin(\omega_p T_s/2)\right)
\]

\[
= G_s(j\omega_p).
\]

That is, a sinusoid with frequency \( \omega_p \) experiences the same transfer function in both continuous and discrete domains.

**Matched Pole-Zero Method**

As it has been presented before, a pole \( s_0 \) of the Laplace transform of a continuous-time signal is related to a pole \( z_0 \) of the z-transform of signal samples according to \( z_0 = e^{s_0 T_s} \).

The idea of zero-pole matching is to use this mapping to determine the location of zeros as well [68]. The procedure is outlined below.

1. A continuous-time pole at \( p = p_0 \) is mapped to a discrete-time pole at \( z = e^{p_0 T_s} \).

2. A continuous-time, finite, zero at \( s = s_0 \) is mapped to a discrete-time zero at \( z = e^{s_0 T_s} \).

3. Let \( N \) and \( M \) be the degree of numerator and denominator of a continuous-time transfer function. If \( N < M \), then the system will have \( (M - N) \) zeros at infinity. Each continuous-time zero at \( s = \infty \) is mapped to a discrete-time zero at \( z = -1 \).

The rationale is that we would like the highest continuous-time frequency of \( s = \infty \) to correspond to the highest discrete-time frequency of \( z = e^{j(\pi/T_s)T_s} = -1 \).

4. Finally, gain of the discrete equivalent is adjusted by making continuous-time and discrete-time systems gains equal at a prespecified frequency \( \omega_0 \), in other words, the following equality is demanded:

\[
G_z(e^{j\omega_0 T_s}) = G_s(j\omega_0)
\]

(3.3.18)

Often in practice the DC gains are required to be equal, that is, \( \omega_0 = 0 \), in which case this should hold:

\[
G_z(1) = G_s(0)
\]

(3.3.19)
Summary

Discretization method choice depends on the application and requirements.

The zero order hold (ZOH) method is well-suited for discrete approximations in the time domain, as it is also called step invariant method. In particular, the step response of the ZOH discretization matches the continuous-time step response at each time step (independently of the sampling rate). Furthermore, both Euler methods (backward and forward differentiation methods) are also time-domain approximations. Backward methods is generally slightly more accurate, however more complex to implement as well.

By contrast, the Tustin and matched methods tend to perform better in the frequency domain because they introduce less gain and phase distortion near the Nyquist frequency, which is half the sampling frequency. For example, the matched method provides more accurate frequency-domain approximation of the notch filter. However, one can further improve the accuracy of the Tustin algorithm by specifying a prewarping frequency at the frequency of interest (for instance, at the notch frequency).

Choice of the sampling rate is yet another important issue that should be addressed here. The higher the sampling rate, the closer the match between the continuous and discretized responses. As a rule of thumb, if one wants the continuous and discretized models to match closely up to some frequency, the Nyquist frequency should be at least two times larger, implying that the sampling frequency should be four times larger. For instance, a filter with notch frequency around $10Hz$, so the sampling frequency should be beyond $40Hz$, which gives a sampling period of $0.025s$.

3.3.2 Standard Digital Filters

As it has been explained before, in case of manual control design, the process consists of the stages depicted in Figure 3.5. As the first stage gathers the required information about the system that is to be controlled, the second one represents a collection of control design techniques, which yield the type and basic parameters of the needed controller and thus the parameters of filters it consists of. Knowing these parameters, a continuous-time transfer function is formed and then, by setting the fixed sampling time, the discrete-time transfer function is derived. In this section, this third stage is elaborated, i.e., a basic derivation of digital filters, knowing their parameters and using discretization methods laid out in the previous section, is presented. See Appendix C for the detailed derivations.

An additional remark should be made here. Four discretization methods, namely Euler backward and forward differentiation and bilinear transformations (Tustin’s and Tustin’s method with prewarping) share the same property – they all transform continuous-time transfer functions $G(s)$ into discrete-time transfer functions $G(z)$ by introducing the transformation $s = f(z)$. Knowing that, an alternative way of discrete-time transfer function derivation can be use, and it is presented in details in Appendix A.

First Order Lowpass

A first lowpass filter is a filter that passes low-frequency signals unscaled, but attenuates, i.e., reduces the amplitude of signals with frequencies higher than the cut-off frequency $f$. In control field, it is mostly used to attenuate noise present at high frequencies.
The continuous-time transfer function of the first order low pass filter is given in Equation (3.3.20). Additionally, Bode plot of it is depicted in Figure 3.10, where the black curve represents the ideal and the red curve a realistic first order lowpass filter.

\[ G(s) = \frac{1}{\frac{1}{2\pi f} s + 1} = \frac{2\pi f}{s + 2\pi f}. \] (3.3.20)

![Figure 3.10: Bode diagram of a first order lowpass filter](image)

The discrete-time transfer functions of the filter, derived using the following discretization methods are:

- **Backward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z-1}{zTs}} G(z) \)
  \[
  G(z) = \frac{1}{\left(\frac{1}{2\pi f T_s} + 1\right) - \frac{1}{2\pi f T_s} z^{-1}}. \] (3.3.21)

- **Forward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z-1}{zTs}} G(z) \)
  \[
  G(z) = \frac{z^{-1}}{2\pi f T_s + (1 - \frac{1}{2\pi f T_s})z^{-1}}. \] (3.3.22)

- **Zero Order Hold (ZOH) Method**, \( G(z) \approx \frac{z^{-1}}{z} Z\left\{\frac{1}{s} G(s)\right\} \)
  \[
  G(z) = \frac{(1 - e^{-2\pi f T_s})z^{-1}}{1 - e^{-2\pi f T_s} z^{-1}}. \] (3.3.23)

- **Tustin’s Method**, \( G(s) \xrightarrow{s \approx \frac{z-1}{\frac{1}{2} + z^{-1}}} G(z) \)
  \[
  G(z) = \frac{1 + z^{-1}}{\left(\frac{1}{2\pi f T_s} + 1\right) + \left(1 - \frac{2}{2\pi f T_s}\right) z^{-1}}. \] (3.3.24)
• Tustin’s method with Prewarping, \( G(s) \xrightarrow{s \approx \alpha \frac{1 - z^{-1}}{1 + z^{-1}}} G(z) \)

\[
\omega_p = 2\pi f \quad \& \quad \alpha = \frac{\omega_p}{\tan \left( \omega_p T_s / 2 \right)}.
\] (3.3.25)

\[
G(z) = \frac{1 + z^{-1}}{(\frac{\alpha}{2\pi f} + 1) + (1 - \frac{\alpha}{2\pi f})z^{-1}}.
\] (3.3.26)

• Matched Pole-Zero Method, \( G(s) \xrightarrow{z_p = e^{\pi p T_s}, z_z = e^{\pi z T_s}} G(z) \)

\[
G(z) = \frac{1 - e^{-2\pi f T_s}z^{-1}}{1 - e^{-2\pi f T_s}z^{-1}}.
\] (3.3.27)

Second Order Lowpass

A second lowpass filter, similarly to the previous one, is a filter that passes low-frequency signals but attenuates the amplitude of signals with frequencies higher than the cut-off frequency \( f \). The difference between the first order lowpass is that it reduces amplitude a lot more after that frequency. It consists of two complex conjugate poles at cut-off frequency with a prescribed damping. The continuous-time transfer function of the second order lowpass filter is given in Equation (3.3.28). Additionally, Bode plot of it is depicted in Figure 3.11, where the black curve represents the ideal and the red curve a realistic second order lowpass filter.

\[
G(s) = \frac{1}{(2\pi f)^2 s^2 + \frac{2D}{2\pi f} s + 1}.
\] (3.3.28)

![Figure 3.11: Bode diagram of a second order lowpass filter](image)

The discrete-time transfer functions of the filter, derived using the following discretization methods are:


3.3. Filter Discretization

- Backward Differentiation Method, \( G(s) \xrightarrow{\frac{s}{z Ts}} G(z) \)

\[
G(z) = \frac{T^2_s}{\left(\frac{2\pi f}{2\pi f - 2s_D} + T_s^2\right) + \left(-\frac{2}{(2\pi f)^2} - \frac{2D_s}{2\pi f T_s}\right)z^{-1} + \frac{1}{(2\pi f)^2}z^{-2}}. \tag{3.3.29}
\]

- Forward Differentiation Method, \( G(s) \xrightarrow{\frac{s}{z Ts}} G(z) \)

\[
G(z) = \frac{T^2_s z^{-2}}{\left(\frac{2\pi f}{2\pi f + 1}\right) + \left(-\frac{2}{(2\pi f)^2} - \frac{2D_s}{2\pi f T_s}\right)z^{-1} + \frac{1}{(2\pi f)^2}z^{-2}}. \tag{3.3.30}
\]

- Zero Order Hold (ZOH) Method, \( G(z) \approx \frac{s}{z} z\left\{G(s)\right\} \)

Derivation and the final formula are left out from the report due to complexity.

- Tustin’s Method, \( G(s) \xrightarrow{\frac{s}{z Ts}} G(z) \)

\[
G(z) = \frac{1 + 2z^{-1} + z^{-2}}{\left(\frac{2\pi f}{2\pi f + 1}\right) + \left(-\frac{2}{(2\pi f)^2} - \frac{2D_s}{2\pi f T_s}\right)z^{-1} + \frac{1}{(2\pi f)^2}z^{-2}}. \tag{3.3.31}
\]

- Tustin’s method with Prewarping, \( G(s) \xrightarrow{\frac{s}{z Ts}} G(z) \)

\[
\omega_p = 2\pi f \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)}, \tag{3.3.32}
\]

\[
G(z) = \frac{1 + 2z^{-1} + z^{-2}}{\left(\frac{2\pi f}{2\pi f + 1}\right) + \left(-\frac{2}{(2\pi f)^2} - \frac{2D_s}{2\pi f T_s}\right)z^{-1} + \frac{1}{(2\pi f)^2}z^{-2}}. \tag{3.3.33}
\]

- Matched Pole-Zero Method, \( G(s) \xrightarrow{z_p = e^{\pi f z_s} z_1 = e^{\pi f z_s}} G(z) \)

\[
G(z) = \frac{(1 - z_{p1} - z_{p2} + z_{p1} z_{p2})}{1 + (-z_{p1} - z_{p2})z^{-1} + z_{p1} z_{p2} z^{-2}}. \tag{3.3.34}
\]

Weak Integrator

A weak integrator is a filter that shares the idea of lowpass filter – it attenuates amplitude of signals below certain frequency. Namely, it also has cut-off frequency \( f \), only it has negative slope before it and passes the signal unscaled beyond that frequency. In control, it is mostly used for achieving zero steady-state error and elimination of low-frequent (constant) disturbances. It consist of a zero at cut-off frequency and a pole at zero frequency \( w = 0 \). The continuous-time transfer function of the weak integrator is given in Equation \( 3.3.35 \).

Additionally, Bode plot of it is depicted in Figure 3.12 where the black curve represents the ideal and the red curve a realistic weak integrator.

\[
G(s) = \frac{s + 2\pi f}{s}. \tag{3.3.35}
\]
The discrete-time transfer functions of the filter, derived using the following discretization methods are:

- **Backward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z-1}{zTs}} G(z) \)
  
  \[
  G(z) = \frac{(1 + 2\pi fT_s z) - z^{-1}}{1 - z^{-1}}. \tag{3.3.36}
  \]

- **Forward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z-1}{zTs}} G(z) \)
  
  \[
  G(z) = \frac{1 + (2\pi fT_s - 1)z^{-1}}{1 - z^{-1}}. \tag{3.3.37}
  \]

- **Zero Order Hold (ZOH) Method**, \( G(z) \approx z^{-1}Z\{\frac{1}{s}G(s)\} \)
  
  \[
  G(z) = \frac{1 + (2\pi fT_s - 1)z^{-1}}{1 - z^{-1}}. \tag{3.3.38}
  \]

- **Tustin’s Method**, \( G(s) \xrightarrow{s \approx \frac{2}{T_s} \frac{1 - z^{-1}}{1 + z^{-1}}} G(z) \)
  
  \[
  G(z) = \frac{(2\pi f + 2\pi f) + (2\pi f - 2\pi f)z^{-1}}{\frac{2}{T_s} - \frac{2}{T_s} z^{-1}}. \tag{3.3.39}
  \]

- **Tustin’s method with Prewarping**, \( G(s) \xrightarrow{s \approx \frac{\omega_p - s^{-1}}{1 + s^{-1}}} G(z) \)
  
  \[
  \omega_p = 2\pi f \quad \& \quad \alpha = \frac{\omega_p}{\tan (\omega_p T_s / 2)}. \tag{3.3.40}
  \]
3.3. Filter Discretization

\[ G(z) = \frac{(\alpha + 2\pi f) + (2\pi f - \alpha)z^{-1}}{\alpha - \alpha z^{-1}}. \]  (3.3.41)

- Matched Pole-Zero Method, \( G(s) \rightarrow G(z) \)

\[ z_p = e^{s_p T_s}, z_z = e^{s_z T_s} \rightarrow G(z) \]

\[ G(z) = \frac{1 - e^{-2\pi f T_s}}{1 - z^{-1}}. \]  (3.3.42)

Lead Lag

The general purpose of a lead lag filter is not to compensate for magnitude, but primarily for phase. Depending on the frequencies of the zero and the pole, the filter can add phase lead \( (f_p > f_z) \) or phase lag \( (f_p < f_z) \). Phase lead archived by a lead lag filter is one of the most commonly used techniques in manual loopshaping, since it improves stability margins in a range of frequencies. The phase maximum is then at frequency \( f_c = \sqrt{f_p f_z} \). The continuous-time transfer function of the lead lag filter is given in Equation (3.3.43).

Additionally, Bode plot of it is depicted in Figure 3.13, where the black curve represents the ideal and the red curve a realistic lead lag filter.

\[ G(s) = \frac{1}{\frac{2\pi f}{2\pi f_p} s + 1}. \]  (3.3.43)

![Figure 3.13: Bode diagram of a lead lag filter](image)

The discrete-time transfer functions of the filter, derived using the following discretization methods are:
Chapter 3. Digital Filters

- Backward Differentiation Method, \( G(s) \xrightarrow{s \approx \frac{z-1}{T_s}} G(z) \)

\[
G(z) = \frac{\left( \frac{1}{2\pi f_s} + T_s \right) - \frac{1}{2\pi f_p} z^{-1}}{\left( \frac{1}{2\pi f_p} + T_s \right) - \frac{1}{2\pi f_p} z^{-1}}.
\] (3.3.44)

- Forward Differentiation Method, \( G(s) \xrightarrow{s \approx \frac{z-1}{T_s}} G(z) \)

\[
G(z) = \frac{\frac{1}{2\pi f_s} z + (1 - \frac{1}{2\pi f_s} T_s) z^{-1}}{\frac{1}{2\pi f_p} z + (1 - \frac{1}{2\pi f_p} T_s) z^{-1}}.
\] (3.3.45)

- Zero Order Hold (ZOH) Method, \( G(z) \approx \frac{z-1}{z} Z\left\{ \frac{1}{\pi} G(s) \right\} \)

\[
G(z) = \frac{\frac{f_p}{f_s} + (1 - \frac{f_p}{f_s} - e^{-2\pi f_p T_s}) z^{-1}}{1 - e^{-2\pi f_p T_s} z^{-1}}.
\] (3.3.46)

- Tustin's Method, \( G(s) \xrightarrow{s \approx \frac{2z-1}{T_s}} G(z) \)

\[
G(z) = \frac{\left( 1 + \frac{1}{2\pi f_s} T_s \right) + (1 - \frac{1}{2\pi f_p} T_s) z^{-1}}{\left( 1 + \frac{1}{2\pi f_p} T_s \right) + (1 - \frac{1}{2\pi f_p} T_s) z^{-1}}.
\] (3.3.47)

- Tustin's method with Prewarping, \( G(s) \xrightarrow{s \approx \alpha \frac{z-1}{1+\frac{z-1}{s}}} G(z) \)

\[
\omega_p = 2\pi \sqrt{f_s f_p} \quad \& \quad \alpha = \frac{\omega_p}{\tan (\omega_p T_s/2)}.
\] (3.3.48)

\[
G(z) = \frac{\left( 1 + \frac{\alpha}{2\pi f_s} \right) + (1 - \frac{\alpha}{2\pi f_p} z^{-1})}{\left( 1 + \frac{\alpha}{2\pi f_p} \right) + (1 - \frac{\alpha}{2\pi f_p} z^{-1})}.
\] (3.3.49)

- Matched Pole-Zero Method, \( G(s) \xrightarrow{z_p=e^{-\pi T_s}, z_z=e^{\pi T_s}} G(z) \)

\[
G(z) = \frac{\left( 1 - e^{-2\pi f_p T_s} - (1 - e^{-2\pi f_p T_s}) e^{-2\pi f_p T_s} z^{-1} \right)}{(1 - e^{-2\pi f_s T_s}) - (1 - e^{-2\pi f_s T_s}) e^{-2\pi f_p T_s} z^{-1}}.
\] (3.3.50)

Skewed Notch

A skewed notch filter is essentially a general second order filter. It has characteristics similar to a lead lag filter. However, in oppose to it, it has narrower phase lead region, yet the phase lead maximum is larger, but the magnitude price that is payed for leading phase is higher. In particular, the magnitude amplification at high frequencies is larger, which is usually unwanted due to presence of noise. The reason for this is that it has two complex conjugate pairs of zeros and poles, instead of a single real zero and pole. The gain at low frequencies is one (zero decibels), while at high frequencies, it equals \( \left( \frac{f_p}{f_s} \right)^2 \). The continuous-time transfer function of the skewed notch filter is given in Equation (3.3.51). Additionally, Bode plot of
it is depicted in Figure 3.14 where the black curve represents the ideal and the red curve a realistic skewed notch filter.

\[ G(s) = \frac{1}{(2\pi f_z)^2} s^2 + \frac{2D}{2\pi f_z} s + 1 \]

\[ \frac{1}{(2\pi f_p)^2} s^2 + \frac{2D}{2\pi f_p} s + 1 \]

(3.3.51)

A special case occurs for \( f_z = f_p \). This filter is no longer skewed and thus it is called only notch filter. The amplitude response of this filter is flat at all frequencies except for the stop band on either side of the center frequency, i.e., a narrow frequency band around the center frequency is rejected, while the rest of the spectrum is left fairly unchanged. The continuous-time transfer function is the same as of a skewed notch filter (3.3.51), while the Bode plot differs. It is depicted in Figure 3.15.

The discrete-time transfer functions of the filter, derived using the following discretization methods are:

- **Backward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{-1}{T}} G(z) \)

\[
G(z) = \left( \frac{1}{(2\pi f_z)^2} + \frac{2D}{2\pi f_z} T_s + T_s^2 \right) + \left( -\frac{2}{(2\pi f_z)^2} - \frac{2D}{2\pi f_z} T_s \right) z^{-1} + \frac{1}{(2\pi f_z)^2} z^{-2}
\]

\[
\left( \frac{1}{(2\pi f_p)^2} + \frac{2D}{2\pi f_p} T_s + T_s^2 \right) + \left( -\frac{2}{(2\pi f_p)^2} - \frac{2D}{2\pi f_p} T_s \right) z^{-1} + \frac{1}{(2\pi f_p)^2} z^{-2}
\]

(3.3.52)

- **Forward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{1}{T}} G(z) \)

\[
G(z) = \left( \frac{1}{(2\pi f_z)^2} + \frac{2D}{2\pi f_z} T_s - \frac{2}{(2\pi f_z)^2} - \frac{2D}{2\pi f_z} T_s + T_s^2 \right) z^{-1} + \frac{1}{(2\pi f_z)^2} z^{-2}
\]

\[
\left( \frac{1}{(2\pi f_p)^2} + \frac{2D}{2\pi f_p} T_s - \frac{2}{(2\pi f_p)^2} - \frac{2D}{2\pi f_p} T_s + T_s^2 \right) z^{-1} + \frac{1}{(2\pi f_p)^2} z^{-2}
\]

(3.3.53)
Chapter 3. Digital Filters

- Zero Order Hold (ZOH) Method, $G(z) \approx \frac{z^{-1}}{z} Z\left\{\frac{1}{s} G(s)\right\}$
  
  Derivation and the final formula are left out from the report due to complexity.

- Tustin’s Method, $G(s) \xrightarrow{s \approx \frac{-1}{z} \frac{1}{1+z^{-1}}} G(z)$
  
  $$G(z) = \frac{\left(\frac{1}{\pi f_z T_z} + \frac{2D_z}{2\pi f_z T_z} + 1\right) \left(\frac{z}{2} - \frac{1}{\pi f_z T_z} + \frac{2D_z}{2\pi f_z T_z} + 1\right)}{\left(\frac{1}{\pi f_z T_z} + \frac{2D_z}{2\pi f_z T_z} + 1\right) + \left(2 - \frac{2}{\pi f_z T_z} + \frac{1}{\pi f_z T_z} - \frac{2D_z}{2\pi f_z T_z} + 1\right)z^{-2}}.$$  
  
  (3.3.54)

- Tustin’s method with Prewarping, $G(s) \xrightarrow{s \approx \frac{-\alpha}{z} \frac{1}{1+z^{-1}}} G(z)$
  
  $$\omega_p = 2\pi \sqrt{f_z} \text{ or } \omega_p = 2\pi \sqrt{f_p} \quad \text{and} \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)}.$$  
  
  (3.3.55)

  $$G(z) = \frac{\left(\frac{\alpha^2}{(2\pi f_z T_z)^2} + \frac{2D_z}{2\pi f_z T_z} + 1\right) \left(\frac{2\alpha^2}{(2\pi f_z T_z)^2} - \frac{2D_z}{2\pi f_z T_z} + 1\right)z^{-1} + \left(\frac{\alpha^2}{(2\pi f_z T_z)^2} - \frac{2D_z}{2\pi f_z T_z} + 1\right)z^{-2}}{\left(\frac{\alpha^2}{(2\pi f_p T_p)^2} + \frac{2D_p}{2\pi f_p T_p} + 1\right) \left(\frac{2\alpha^2}{(2\pi f_p T_p)^2} - \frac{2D_p}{2\pi f_p T_p} + 1\right)z^{-1} + \left(\frac{\alpha^2}{(2\pi f_p T_p)^2} - \frac{2D_p}{2\pi f_p T_p} + 1\right)z^{-2}}.$$  
  
  (3.3.56)

- Matched Pole-Zero Method, $G(s) \xrightarrow{z \approx e^{sT_s}, z \approx e^{sT_s}} G(z)$
  
  $$G(z) = \frac{(1 - z_{p1} - z_{p2} + z_{p1} z_{p2}) \left(1 + (-z_{z1} - z_{z2})z^{-1} + z_{z1} z_{z2} z^{-2}\right)}{(1 - z_{z1} - z_{z2} + z_{z1} z_{z2}) \left(1 + (-z_{p1} - z_{p2})z^{-1} + z_{p1} z_{p2} z^{-2}\right)}.$$  
  
  (3.3.57)

**PD**

The Proportional-Derivative (PD) filter is widely used in control, in particular for fast response controllers that do not need a steady-state error equals to zero. Since both propor-
3.3. Filter Discretization

Proportional and derivative controllers are fast, the two together yield convergence even faster. Proportional action provides an instantaneous response to the control error, which is useful for improving the response of a stable system but cannot control an unstable system by itself. Additionally, the gain is the same for all frequencies leaving the system with a nonzero steady-state error. Derivative action, on the other hand, acts on the derivative or rate of change of the control error. This provides a fast response, but cannot accommodate elimination of constant errors – the derivative of a constant, but still nonzero error is 0. The derivative control produces large control signals in response to high frequency control errors such as set point changes (step command) and measurement noise [25]. The filter consists of only one zero at $s_z = -\frac{k_p}{k_v}$ and has no poles, i.e., the transfer function is non-proper. That implies that some of the discretization methods are not feasible since they yield non-causal discrete system, which cannot be implemented by the chosen realization technique.

The continuous-time transfer function of the PD filter is given in Equation (3.3.58). Additionally, Bode plot of it is depicted in Figure 3.16 where the black curve represents the ideal and the red curve a realistic PD filter.

$$G(s) = k_p + k_v s. \quad \text{(3.3.58)}$$

![Figure 3.16: Bode diagram of a PD filter](image)

The discrete-time transfer functions of the filter, derived using the following discretization methods are:

- **Backward Differentiation Method**, $G(s) \xrightarrow{s \approx \frac{z-1}{T_s}} G(z)$

  $$G(z) = \frac{(k_p T_s + k_v) - k_v z^{-1}}{T_s}. \quad \text{(3.3.59)}$$

- **Forward Differentiation Method**, $G(s) \xrightarrow{s \approx \frac{z-1}{T_s}} G(z)$
Not possible to implement in Transposed-Direct-Form II (TDF-II), due to no poles in the continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.

- **Zero Order Hold (ZOH) Method**, \( G(z) \approx \frac{1}{z} \mathcal{Z}\left\{ \frac{1}{s} G(s) \right\} \)

\[
G(z) = (k_v + k_p) - k_v z^{-1}. \tag{3.3.60}
\]

- **Tustin's Method**, \( G(s) \xrightarrow{s \approx \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}}} G(z) \)

\[
G(z) = \frac{(k_p + k_v \frac{2}{T_s}) + (k_p - k_v \frac{2}{T_s}) z^{-1}}{1 + z^{-1}}. \tag{3.3.61}
\]

- **Tustin's method with Prewarping**, \( G(s) \xrightarrow{s \approx \omega_p \frac{1-z^{-1}}{1+z^{-1}}} G(z) \)

\[
\omega_p = 2\pi \frac{k_p}{k_v} \quad \alpha = \frac{\omega_p}{\tan (\omega_p T_s/2)}
\]

\[
G(z) = \frac{(k_p + k_v \alpha) + (k_p - k_v \alpha) z^{-1}}{1 + z^{-1}}. \tag{3.3.63}
\]

- **Matched Pole-Zero Method**, \( G(s) \xrightarrow{z_p = e^{s_p T_s}, s_z = e^{s_z T_s}} G(z) \)

Not possible to implement in Transposed-Direct-Form II (TDF-II), due to no poles in the continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.

In order to overcome implementation problems of derivative action (having solely a zero in the transfer function), improve performance in the real-time usage and avoid the fact that any abrupt change in set point would lead into extremely large control signal, there have been several suggestions in literature [64]. The one that has shown the best results is introduction of a pole, i.e., a filter coefficient \( N \), which determines the pole location of the filter in the derivative action. Now the continuous-time transfer function of a PD filter becomes:

\[
G(s) = k_p + k_v \frac{N s}{s - N}. \tag{3.3.64}
\]

Transfer function is basically the same as the one of a lead lag filter. However, one wants to keep factor \( N \) high, since (nonrealistic) case \( N \to \infty \) would converge to the textbook version of PD filter, see [3.3.58]. This is derived from the block diagram depicted in Figure 3.17.

The derivations of PD filter discrete-time transfer functions with filter coefficient \( N \) using different discretization methods are presented below.

- **Improved Backward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z-1}{z^{-1}}} G(z) \)

\[
G(z) = \frac{(k_p - k_p NT_s + k_v N) + (-k_p - k_v N) z^{-1}}{(1 - NT_s) - z^{-1}}. \tag{3.3.65}
\]
3.3 Filter Discretization

Figure 3.17: Block diagram of a PD filter with a pole added to the derivative action

- Improved Forward Differentiation Method, \( G(s) \xrightarrow{\frac{z-1}{z}} G(z) \)
  \[
  G(z) = \frac{(k_p + k_v N) + (-k_p - k_p NT_s - k_d N)z^{-1}}{1 + (-1 - NT_s)z^{-1}}. \tag{3.3.66}
  \]

- Improved Zero Order Hold (ZOH) Method, \( G(z) \approx \frac{z-1}{z}Z\{\frac{1}{z}G(s)\} \)
  \[
  G(z) = \frac{(k_p + k_v N) + (-k_p e^{NT_s} - k_u N)z^{-1}}{1 - e^{NT_s}z^{-1}}. \tag{3.3.67}
  \]

- Improved Tustin’s Method, \( G(s) \xrightarrow{\frac{z-1}{z}} G(z) \)
  \[
  G(z) = \frac{(k_p \frac{2}{Ts} - k_p N + k_v N \frac{2}{Ts}) + (-k_p \frac{2}{Ts} - k_p N - k_v N \frac{2}{Ts})z^{-1}}{(\frac{2}{Ts} - N) + (-\frac{2}{Ts} - N)z^{-1}}. \tag{3.3.68}
  \]

- Improved Tustin’s method with Prewarping, \( G(s) \xrightarrow{\frac{z-1}{z}} G(z) \)
  \[
  \omega_p = 2\pi \frac{k_p}{k_v} \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p Ts/2)}. \tag{3.3.69}
  \]
  \[
  G(z) = \frac{(k_p \alpha - k_p N + k_v N \alpha) + (-k_p \alpha - k_p N - k_v N \alpha)z^{-1}}{(\alpha - N) + (-\alpha - N)z^{-1}}. \tag{3.3.70}
  \]

- Improved Matched Pole-Zero Method, \( G(s) \xrightarrow{e^{sTs}, z = e^{sTs}} G(z) \)
  \[
  G(z) = k_p \frac{1 - e^{NT_s}}{1 - e^{k_p N + k_v N}T_s z^{-1}} \tag{3.3.71}
  \]

PID

Proportional-Integral-Derivative (PID) filter is also widely used in control, and is essentially just an extension of PD filter. The difference is introduced by the integral term, which, as the name suggests, represents integral of the error (input). In the context of control, the integral term accelerates the movement of the process towards set point and eliminates the
residual steady-state error that occurs with a PD controller. However, since it responds to accumulated errors from the past, it can cause the present value to overshoot the set point value and make system unstable. This is accompanied usually with a phenomenon called “windup”. Integral windup refers to the situation where the integral term accumulates a significant error during the rise (windup) [59]. This topic is thoroughly treated and a solution is offered in Appendix B.

The filter consists of only one pole at 0 frequency and two zeros, i.e., transfer function is non-proper. That implies that some of the discretization methods are not feasible with the chosen realization technique.

The continuous-time transfer function of the PID filter is given in Equation (3.3.72). Additionally, Bode plot of it is depicted in Figure 3.18 where the black curve represents the ideal and the red curve a realistic PID filter.

\[
G(s) = k_p + k_v s + \frac{k_i}{s} = \frac{k_p s + k_v s^2 + k_i}{s}. \tag{3.3.72}
\]

![Bode diagram of a PID filter](image)

The discrete-time transfer functions of the filter, derived using the following discretization methods are:

- **Backward Differentiation Method**, \(G(s) \xrightarrow{s \approx \frac{s - 1}{T_s}} G(z)\)

\[
G(z) = \frac{(k_p T_s + k_v + k_i T_s^2) + (-k_p T_s - 2k_v)z^{-1} + k_v z^{-2}}{T_s - T_s z^{-1}}. \tag{3.3.73}
\]

- **Forward Differentiation Method**, \(G(s) \xrightarrow{s \approx \frac{s - 1}{T_s}} G(z)\)

Not possible to implement in Transposed-Direct-Form II (TDF-II), due to a non-proper continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.
3.3. Filter Discretization

- **Zero Order Hold (ZOH) Method**, \(G(z) \approx \frac{z^{-1}}{z} \mathcal{Z}\{\frac{1}{s} G(s)\}\)

\[
G(z) = \frac{(k_v + k_p) + (k_i T_s - 2k_v - k_p)z^{-1} + k_v z^{-2}}{1 - z^{-1}}. \quad (3.3.74)
\]

- **Tustin’s Method**, \(s \approx \frac{2}{T_s} \frac{1 + z^{-1}}{1 - z^{-1}} \rightarrow G(z)\)

\[
G(z) = \frac{(k_p \frac{2}{T_s} + k_v \frac{2}{T_s} + k_i) + (2k_i - 2k_v \frac{2}{T_s})z^{-1} + (k_i \frac{2}{T_s} + k_i - k_p \frac{2}{T_s})z^{-2}}{\frac{2}{T_s} - 2 \frac{2}{T_s} z^{-2}}. \quad (3.3.75)
\]

- **Tustin’s method with Prewarping**, \(s \approx \alpha \frac{1 + z^{-1}}{1 - z^{-1}} \rightarrow G(z)\)

\[
\omega_p = 2\pi \frac{-k_v - \sqrt{k_v^2 - 4k_i k_p}}{2k_p} \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)}. \quad (3.3.76)
\]

\[
G(z) = \frac{(k_p \alpha + k_i \alpha + k_i) + (2k_i - 2k_v \alpha)z^{-1} + (k_i \alpha + k_i - k_p \alpha)z^{-2}}{\alpha - \alpha z^{-2}}. \quad (3.3.77)
\]

- **Matched Pole-Zero Method**, \(G(s) \xrightarrow{z_p = e^{s p T_s}, z_z = e^{s z T_s}} G(z)\)

Not possible to implement in Transposed-Direct-Form II (TDF-II), due to a non-proper continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.

Again, the same holds as for the PD filter. In order to overcome implementation problems (having a non-proper transfer function), improve performance in the real-time usage and avoid the fact that any abrupt change in set point would lead to an extremely large control signal, there have been several suggestions in literature [64]. The one that has shown the best results is the introduction of a pole, i.e., filter coefficient \(N\), which determines the pole location of the filter in the derivative action. Now the continuous-time transfer function of a PID filter becomes:

\[
G(s) = k_p + k_v \frac{Ns}{s + N} + k_i \frac{1}{s} = \frac{(k_p + k_i N)s^2 + (k_p N + k_i) s + k_i N}{s^2 + Ns}. \quad (3.3.78)
\]

One wants to keep factor \(N\) high, since (nonrealistic) case \(N \rightarrow \infty\) would converge to the textbook version of PD filter, see [3.3.72]. This is derived from the block diagram depicted in Figure 3.19.

The derivations of PID filter discrete-time transfer functions with filter coefficient \(N\) using different discretization methods are presented below:

- **Improved Backward Differentiation Method**, \(G(s) \xrightarrow{s \approx \frac{z^{-1}}{T_s}} G(z)\)
Chapter 3. Digital Filters

Figure 3.19: Block diagram of a PID filter with added pole to derivative action

\[
G(z) = \frac{(k_p + k_v N + k_p N T_s + k_i T_s + k_i N T_s^2) + \cdots}{(1 + N T_s) + (-2 - N T_s) z^{-1} + z^{-2}}
\]

\[
\cdots + \frac{(-2 k_p - 2 k_i N - k_i T_s - k_p N T_s) z^{-1} + \cdots}{(1 + N T_s) + (-2 - N T_s) z^{-1} + z^{-2}}
\]

\[
\cdots + \frac{(k_p + k_v N) z^{-2}}{(1 + N T_s) + (-2 - N T_s) z^{-1} + z^{-2}}.
\]

(3.3.79)

- Improved Forward Differentiation Method, \(G(s) \xrightarrow{s \rightarrow \frac{z-1}{T_s}} G(z)\)

\[
G(z) = \frac{(k_p + k_v N) + \cdots}{1 + (-2 + N T_s) z^{-1} + (1 - N T_s) z^{-2}}
\]

\[
\cdots + \frac{(-2 k_p - 2 k_i N + k_p N T_s + k_i T_s) z^{-1} + \cdots}{1 + (-2 + N T_s) z^{-1} + (1 - N T_s) z^{-2}}
\]

\[
\cdots + \frac{(k_p + k_v N - k_p N T_s - k_i T_s + k_i N T_s^2) z^{-2}}{1 + (-2 + N T_s) z^{-1} + (1 - N T_s) z^{-2}}.
\]

(3.3.80)

- Improved Zero Order Hold (ZOH) Method, \(G(z) \approx \frac{z-1}{z} \mathcal{Z}\{\frac{1}{s} G(s)\}\)

\[
G(z) = \frac{(k_p + k_v N) + (-k_p - k_p e^{-N T_s} - 2 k_i N + k_i T_s) z^{-1} + \cdots}{1 + (-1 - e^{-N T_s}) z^{-1} + e^{-N T_s} z^{-2}}
\]

\[
\cdots + \frac{(k_p e^{-N T_s} + k_v N - k_i T_s e^{-N T_s}) z^{-2}}{1 + (-1 - e^{-N T_s}) z^{-1} + e^{-N T_s} z^{-2}}.
\]

(3.3.81)

- Improved Tustin’s Method, \(G(s) \xrightarrow{s \rightarrow \frac{z-1}{z} + \frac{T_s}{2}} G(z)\)

52
3.4. Filter Utilization Example

In this section, an example of a digital filter implementation is presented. The lead lag filter is chosen and it is utilized through the OROCOS© environment. In particular, a lead lag filter component is created in OROCOS©, relying on the S&C Library component (for the code of the lead lag digital filter component see Appendix[D]). The goal of the example is to show how the digital filter works and the ease of using the library to obtain a digital filter that can be used for real-time control.

\[ G(z) = \frac{((k_p + k_v N) \left(\frac{z}{T_s}\right)^2 + (k_p N + k_i) \frac{z}{T_s} + k_i N) + \cdots}{((\frac{z}{T_s})^2 + N \frac{z}{T_s}) - 2(\frac{z}{T_s})^2 z^{-1} + ((\frac{z}{T_s})^2 - N \frac{z}{T_s}) z^{-2} + \cdots} \]

\[ = \frac{((2k_p - 2k_v N) \left(\frac{z}{T_s}\right)^2 + 2k_i N) z^{-1} + \cdots}{((\frac{z}{T_s})^2 + N \frac{z}{T_s}) - 2(\frac{z}{T_s})^2 z^{-1} + ((\frac{z}{T_s})^2 - N \frac{z}{T_s}) z^{-2} + \cdots} \]

\[ = \frac{((k_p + k_v N) \left(\frac{z}{T_s}\right)^2 + (-k_p N - k_i) \frac{z}{T_s} + k_i N) z^{-2}}{((\frac{z}{T_s})^2 + N \frac{z}{T_s}) - 2(\frac{z}{T_s})^2 z^{-1} + ((\frac{z}{T_s})^2 - N \frac{z}{T_s}) z^{-2} + \cdots}. \]

(3.3.82)

- Improved Tustin’s method with Prewarping, \( G(s) \xrightarrow{s \approx \frac{1-z^{-1}}{1+z^{-1}}} G(z) \)

\[ \omega_p = 2\pi \left( \frac{-k_v - \sqrt{k_v^2 - 4k_i k_p}}{2k_p} \right) \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)}. \]  

(3.3.83)

\[ G(z) = \frac{((k_p + k_v N) \alpha^2 + (k_p N + k_i) \alpha + k_i N) + \cdots}{(\alpha^2 + N\alpha) - 2\alpha^2 z^{-1} + (\alpha^2 - N\alpha) z^{-2} + \cdots} \]

\[ + \frac{((2k_p - 2k_v N) \alpha^2 + 2k_i N) z^{-1} + \cdots}{(\alpha^2 + N\alpha) - 2\alpha^2 z^{-1} + (\alpha^2 - N\alpha) z^{-2} + \cdots} \]

\[ + \frac{((k_p + k_v N) \alpha^2 + (-k_p N - k_i) \alpha + k_i N) z^{-2}}{(\alpha^2 + N\alpha) - 2\alpha^2 z^{-1} + (\alpha^2 - N\alpha) z^{-2} + \cdots}. \]

(3.3.84)

- Improved Matched Pole-Zero Method, \( G(s) \xrightarrow{s_p = e^{s_p T_s}, z_s = e^{s_s T_s}} G(z) \)

First, poles \( s_{p1,2} \) and zeros \( s_{z1,2} \) of a continuous-time transfer function need to be found. They are equal to the roots of a denominator and numerator, respectively:

\[ s_{p1} = 0 \quad \& \quad s_{p2} = -N. \]  

(3.3.85)

\[ s_{z1,2} = \frac{-k_p N - k_i \pm \sqrt{(k_i + k_p N)^2 - 4k_i(k_p + k_v N)}}{2(k_p + k_v N)}. \]

(3.3.86)

Now, these zeros and poles are mapped into discrete-time domain \( z_{s1,2} = e^{s_{s1,2} T_s} \) and \( z_{p1} = 1, \quad z_{p2} = e^{-N T_s} \), respectively.

\[ G(z) = \frac{1 + (-z_{z1} - z_{z2}) z^{-1} + z_{z1} z_{z2} z^{-2}}{1 + (-z_{p1} - z_{p2}) z^{-1} + z_{p1} z_{p2} z^{-2}}. \]  

(3.3.87)
An important characteristic of the digital filters is their frequency response functions (FRFs). Therefore, a simulation scheme that can analyze these is created and it is depicted in Figure 3.20. A Gaussian noise signal is used as an input to the lead lag filter and it is recorded in the reporter. The reporter is an OROCOS® component that stores all the input signals into a text file, that can be later analyzed. In this case, it also saves the values that are filtered through the filter. The filter has a configuration parameter by which a discretization method can be changed. Having the input and the output of the filter saved, for different discretization methods, one can proceed with data analysis.

To estimate the frequency response of the filter, the MATLAB™ function `tfestimate` is used. Feeding the function with input and output of the system, it returns a frequency response of it at the given frequencies (which were chosen to cover the whole range of interest). The obtained results are shown in Figure 3.21.

The concrete parameters of the lead lag filter used are: zero frequency $f_z = 4\text{Hz}$, pole frequency $f_p = 36\text{Hz}$ and sampling time $0.005\text{s}$ (equivalent to the sampling frequency $200\text{Hz}$). These parameters comprise a lead filter with a peak of phase lead at $\sqrt{f_z \cdot f_p} = 12\text{Hz}$. The six frequency response functions, corresponding to the six differ-
ent discretization methods, try to reassemble the ideal lead lag filter, that is represented in Figure 3.21 with the black curve. One can see that all the discretizations yield a response that does not deviate a lot from the ideal one. Some match better phase characteristic and the other ones the magnitude characteristic. Tustin method with prewarping is chosen to be the default one, however the user is able to change this at the moment of creation of the component, as well as at any other point of usage.
Chapter 4

Kinematic Models of Wheeled Mobile Robots

Wheeled mobile robots (WMR) are increasingly present in service and industrial robotics, especially when flexible motion abilities on relatively smooth surfaces are required \[54\]. WMR are wheeled vehicles which are capable of autonomous motion. They comprise a class of mechanical systems characterized by kinematic constraints that are not possible to integrate and hence that cannot be eliminated from the model equations, unlike, e.g., robotic manipulators. This implies that the standard planning and control algorithms developed for those manipulators without constraints are no more applicable \[17\]. Despite a vast amount of literature dealing with these algorithms for specific, simplified kinematic models of rigid WMR, like for instance a systematic procedure for model derivation (see \[40\]), most of today’s wheeled mobile robots have generally a much more complex constructive structure (e.g., robots with three or four omniwheels) and for which the modeling issue, which conditions motion planning and control design, is still a significant question.

However, the work of Campion, Bastin and D’Andrea-Novel \[17\] suggests a solution and tries to cope with this issue. It consists of modeling of a general WMR with the structural properties of the kinematic and dynamic models, taking into account the restriction to the robot mobility induced by the physical constraints and with an arbitrary number of wheels of various types and various motorizations, together with partitioning of the set of WMR into five classes. The purpose of this part of the project is to integrate a part of it, i.e., the parameterized geometric transformation between joint (wheel) space and Euclidian space for some of the mobile platform classes in the Systems and Control Library and for that, the required theory is presented in this chapter. The transformation would enable control of the WMR as control of multiple Single-Input Single-Output (SISO) systems (for the motivation see, e.g., \[52\]).

Firstly, the definition of robot posture is given and general terminology is introduced. That is followed by the description of wheels which can form a wheeled mobile platform. Next, the definition of the kinematic families is given. This comprises the identification of restrictions to the robot mobility, definition of degrees of mobility and steerability and classification with respect to those degrees. The second section offers slightly different classification, in terms of implementation. Finally, after introducing all the needed the-
4.1 Robot Posture

The description of the position of the robot on the plane is presented in Figure 4.2. An arbitrary orthonormal inertial basis \( \{ \vec{K}, \vec{M} \} \) is fixed in the plane of the motion. An arbitrary reference point \( C \) (usually center of gravity of the robot) on the frame and an arbitrary basis \( \{ \vec{x}_1, \vec{x}_2 \} \) attached to the frame are defined. The position of the robot is then completely specified by the 3 variables \( x, y, \theta \). \( \theta \) is the orientation of the basis \( \{ \vec{x}_1, \vec{x}_2 \} \) with respect to the inertial basis \( \{ 0, \vec{K}, \vec{M} \} \) and \( x, y \) are the coordinates of the reference point \( C \) in the inertial basis given as

\[
\vec{OC} = x \vec{K} + y \vec{M}.
\]  

(4.1.1)

The robot position is described by vector \( \xi \):

\[
\xi \triangleq \begin{pmatrix} x \\ y \\ \theta \end{pmatrix}.
\]  

(4.1.2)

Additionally, the orthogonal rotation matrix is defined:

\[
R(\theta) \triangleq \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix}.
\]  

(4.1.3)
4.2 Description of Wheels

In order to describe wheels, a set of assumptions needs to be introduced. First, during the motion, the plane of each wheel remains vertical and the wheel rotates around its horizontal axle. The horizontal axle can have fixed or varying orientation with respect to the frame $C$. Moreover, the contact between the wheel and the ground is reduced to a single point of the plane. In that case, two basic classes of idealized wheels can be distinguished: the conventional wheels and the Swedish wheels [55]. For each of them, additional assumptions are made:

- For a conventional wheel, the contact between the wheel and the ground is supposed to satisfy the pure rolling without slipping condition. This means that the velocity of the contact point is equal to zero and implies that the components of this velocity parallel and orthogonal to the plane of the wheel are equal to zero.

- For a Swedish wheel, only one component of the velocity of the contact point of the wheel with the ground is supposed to be equal to zero along the motion. The direction of this zero component of the velocity is arbitrary a priori but is fixed with respect to the orientation of the wheel.

Based upon these assumptions, explicit expressions of the constraints for conventional and Swedish wheels can be derived. This allows the definition of the total constraints of a WMR, depending on the wheel configuration, and by that the derivation of a solver for the kinematic model of that WMR.

4.2.1 Conventional Wheels

There are several types of conventional wheels, namely fixed wheels, centered orientable wheels and off-centered orientable wheels [65]. Examples are depicted in Figure 4.3

Fixed wheels

In Figure 4.4 the configuration of a fixed wheel is depicted. $A$ denotes center of the wheel and a fixed point of the frame. Its position in the basis $\{\vec{x}_1, \vec{x}_2\}$ is represented...
using polar coordinates, \(i.e.\), the distance \(CA = l\) and the angle \(\alpha\). The constant angle \(\beta\) corresponds to the orientation of the plane of the wheel with respect to \(CA\), the rotation angle of the wheel around its (horizontal) axle is denoted \(\varphi\), while \(r\) is the radius of the wheel.

Hence, the four constants \(\alpha\), \(\beta\), \(l\) and \(r\) characterize the position of the wheel, whilst the motion is represented by a time varying angle \(\varphi\). Out of this, constraints along and orthogonal to the wheel plane can be derived.

\[
\begin{bmatrix} -\sin(\alpha + \beta) & \cos(\alpha + \beta) & l \cos \beta \end{bmatrix} R(\theta) \dot{\xi} + r \dot{\varphi} = 0 \quad (4.2.1)
\]

\[
\begin{bmatrix} \cos(\alpha + \beta) & \sin(\alpha + \beta) & l \sin \beta \end{bmatrix} R(\theta) \dot{\xi} = 0. \quad (4.2.2)
\]

### Centered orientable wheels

The motion of the wheel plane with respect to the frame of a centered orientable wheel is a rotation around a vertical axle passing through the center of it, see again Figure 4.4. The description is the same as for a fixed wheel, except that now the angle \(\beta\) is time-varying and not constant. The wheel position is represented by three constants \(\alpha\), \(l\) and \(r\) and the motion with respect to the frame by two time-varying angles \(\beta\) and \(\varphi\). The constraints have the same form as in the case of fixed wheels, see 4.2.1 and 4.2.2.
4.2. Description of Wheels

Off-centered orientable wheels

This type of wheels, also known as "castor" wheels, is also orientable with respect to the frame. However, the rotation of the wheel plain is not around the center of the wheels, but a vertical axle which does not pass through it, see Figure 4.5. Hence, additional parameters are needed to describe the position and the motion of the wheel. The two constraints are of the following form:

\[
\begin{align*}
-\sin (\alpha + \beta) \cos (\alpha + \beta) & \quad l \cos \beta \quad R(\theta) \dot{\xi} + r \dot{\varphi} = 0 \quad (4.2.3) \\
\cos (\alpha + \beta) \sin (\alpha + \beta) & \quad d + l \sin \beta \quad R(\theta) \dot{\xi} + d \dot{\beta} = 0, \quad (4.2.4)
\end{align*}
\]

where \(\alpha\), \(l\), \(r\) and \(d\) represent the position and \(\beta\) and \(\varphi\) the motion of the wheel. The center of the wheel is now denoted \(B\) and is connected to the frame by a rigid bar \(AB\) of constant length \(d\) which can rotate around a fixed vertical axle at point \(A\). \(A\) is a fixed point of the frame and its position is specified by the two polar coordinates as in the fixed wheel case. The plane of the wheel is aligned along \(AB\).

![Figure 4.5: Conventional off-centered orientable wheels](image)

4.2.2 Swedish Wheels

The position of the Swedish wheel is characterized by the three constant parameters \(\alpha\), \(\beta\) and \(l\), which is the same as for the fixed wheel [17]. However, an additional parameter \(\gamma\) is needed to represent the direction with respect to the wheel plane of the zero component of the velocity of the contact point, see Figure 4.6. The motion constraint is subsequently expressed as follows:

\[
\begin{align*}
-\sin (\alpha + \beta + \gamma) \cos (\alpha + \beta + \gamma) & \quad l \cos (\beta + \gamma) \quad R(\theta) \dot{\xi} + r \cos \gamma \dot{\varphi} = 0 \quad (4.2.5)
\end{align*}
\]

Two different types are most common, namely Mecanum wheels and omni wheels [69]. The former type is a conventional wheel with a series of rollers attached to its circumference with a certain angle, while the latter one is a wheel with small discs around the circumference which are perpendicular to the rolling direction. Both wheel configurations are omnidirectional, i.e., they achieve traction in one direction and allows passive motion in another direction. Examples of both are shown in Figure 4.7.
4.3 Definition of Kinematic Families

Defining wheels, together with constraints they impose, allows their combination with the goal to form a wheeled mobile platform. The main objective of this section is to model the kinematics of such a WMR and single out general characteristics based on which division into families or classes can be done.

A wheeled mobile robot can be comprehensively defined as a wheeled vehicle equipped with embedded motors and a computer by which it is controlled and as such, it is capable of autonomous motion and generally, kinematics are defined as a study of the geometry of motion [40]. Before modeling of the kinematics can be performed, it is important to determine a certain set of premises with respect to WMR. It is assumed that the mobile robots are built of a rigid frame supplied with wheels that are not deformable and which are moving on a horizontal plane. The additional assumptions are elaborated in the following sections.

4.3.1 Restrictions to the Robot Mobility

As suggested in [17], restrictions to the mobility of the robot depending on its wheel configuration are imposed. For the sake of universality, a general mobile robot is considered. Moreover, for the ease of exposition, the following subscripts to identify quantities relative to these four classes are introduced: \( f \) for conventional fixed wheels, \( c \) for conventional centered orientable wheels, \( oc \) for conventional off-centered orientable wheels, and \( sw \) for Swedish wheels. The numbers of wheels of each type are denoted \( N_f, N_c, \)
4.3. Definition of Kinematic Families

Now, the robot configuration can be entirely described using the following vectors of coordinates.

1. Posture coordinates
   Coordinates which represent the position coordinates in the plane, see Equation (4.1.2).

2. Angular coordinates
   These coordinates are only present in the case of conventional center and off-center wheels. They are represented by the orientation angles $\beta_c(t)$ and $\beta_{oc}(t)$, respectively.

3. Rotation coordinates
   Coordinates which characterize the rotation angles of the wheels around their horizontal axle of rotation.

Collectively, the coordinates $\xi$, $\beta_c$, $\beta_{oc}$, $\varphi$ are called the set of configuration coordinates.

This leads to the total number of configuration coordinates, $N_f + 2N_c + 2N_{oc} + N_{sw} + 3$ (3 originates from posture coordinates).

With the notations exposed above, the constraints can be written under the general matrix form:

$$J_1(\beta_c, \beta_{oc})R(\theta)\dot{\xi} + J_2\dot{\varphi} = 0$$  \hspace{1cm} (4.3.2)

$$C_1(\beta_c, \beta_{oc})R(\theta)\dot{\xi} + C_2\dot{\beta}_{oc} = 0,$$  \hspace{1cm} (4.3.3)

with

$$J_1(\beta_c, \beta_{oc}) \triangleq \begin{pmatrix} J_{1f} & J_{1c}(\beta_c) & J_{1oc}(\beta_{oc}) \\ \end{pmatrix}$$

$$C_1(\beta_c, \beta_{oc}) \triangleq \begin{pmatrix} C_{1f} & C_{1c}(\beta_c) & C_{1oc}(\beta_{oc}) \\ \end{pmatrix}$$

$C_2$ is a diagonal matrix

where $J_{1f}, J_{1c}(\beta_c), J_{1oc}(\beta_{oc}), J_{1sw}$ are matrices which form is derived directly from Equations (4.2.1), (4.2.3) and (4.2.5). $J_{1f}$ and $J_{1sw}$ are constant, while $J_{1c}$ and $J_{1oc}$ are time varying respectively through $\beta_c$ and $\beta_{oc}$. $J_2$ is a constant matrix which diagonal entries are the radii of the wheels, with the exception of the radii of the Swedish wheels which are multiplied by $\cos \gamma$. $C_{1f}, C_{1c}(\beta_c)$ and $C_{1oc}(\beta_{oc})$ are 3 matrices which rows are derived from the constraints (4.2.2) and (4.2.4). $C_{1oc}(\beta_{oc})$ is constant, while $C_{1f}(\beta_c)$ and $C_{1sw}(\beta_{oc})$ are time-varying with respect to the angular coordinates. $C_{2oc}$ is a diagonal matrix which diagonal entries equal parameter $d$ for the $N_{oc}$ off-centered orientable wheels.

Here, a remark should be made regarding the Swedish wheels. The angle $\gamma$ should never be equal to $\pi/2$, since this would correspond to the direction of the zero component of
the velocity being orthogonal to the plane of the wheel. By that, the advantage of this wheel would be lost and it would behave as conventional wheel.

An example of the constraints of a WMR with four omni wheels, configured with the parameters in Table 4.1 is given. Omni wheels of the considered WMR are forming a square, as shown in Figure 4.8. In particular, that means that all the $\alpha$ angles are equal ($\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \alpha$) and that all the $l$ distances of the wheels from the center of gravity are equidistant ($l_1 = l_2 = l_3 = l_4 = L$). The radius of all the wheels is the same, $r$.

**Table 4.1:** Parameters of the considered WMR with four omni wheels

<table>
<thead>
<tr>
<th>Wheel</th>
<th>$\alpha$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>$l$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omni wheel 1</td>
<td>$\alpha_1$</td>
<td>0</td>
<td>0</td>
<td>$l_1$</td>
</tr>
<tr>
<td>Omni wheel 2</td>
<td>$\alpha_2$</td>
<td>0</td>
<td>0</td>
<td>$l_2$</td>
</tr>
<tr>
<td>Omni wheel 3</td>
<td>$\alpha_3$</td>
<td>0</td>
<td>0</td>
<td>$l_3$</td>
</tr>
<tr>
<td>Omni wheel 4</td>
<td>$\alpha_4$</td>
<td>0</td>
<td>0</td>
<td>$l_4$</td>
</tr>
</tbody>
</table>

The resulting constraint matrices $J_1$ and $J_2$ are shown below. Since the WMR consists of only omni wheels, the constraints $C_1$ and $C_2$ do not exist.

$$J_1(\beta_c, \beta_{oc}) = J_1 = \begin{pmatrix} -\sin \alpha_1 & \cos \alpha_1 & l_1 \\ -\sin \alpha_2 & \cos \alpha_2 & l_2 \\ -\sin \alpha_3 & \cos \alpha_3 & l_3 \\ -\sin \alpha_4 & \cos \alpha_4 & l_4 \end{pmatrix}, \quad J_2 = \text{diag}(r)$$

![Figure 4.8: Example of (3, 0) mobile platform (left) and (2, 0) mobile platform (right)](image)

### 4.3.2 Degree of Mobility and Degree of Steerability

If only $N_f$ and $N_s$ constraints from Equation (4.3.3) are considered, they can be written as in Equation (4.3.4).

$$C_{1_f} R(\theta) \dot{\xi} = 0, \quad C_{1_s} (\dot{\beta}_c) R(\theta) \dot{\xi} = 0. \quad (4.3.4)$$
4.3. Definition of Kinematic Families

From these constraints it yields that the vector $R(\theta)\dot{\xi}$ belongs to the null space of the matrix $C_1^*(\beta_c)$:

$$C_1^*(\beta_c) = \begin{pmatrix} C_{1f} \\ C_{1c}(\beta_c) \end{pmatrix}.$$  \hfill (4.3.5)

This can also be expressed as:

$$R(\theta)\dot{\xi} \in N(C_1^*(\beta_c)).$$  \hfill (4.3.6)

It implies that rank $[C_1^*(\beta_c)] \leq 3$. In case rank $[C_1^*(\beta_c)] = 3$, then $R(\theta)\dot{\xi} = 0$ and no motion in the plane is possible. In other words, the limitations of the robot mobility are directly related to the rank of $C_1^*$.

The design of the mobile robot and the wheel configuration determines the rank of matrix $C_1^*(\beta_c)$. Therefore, the degree of mobility $\delta_m$ of a mobile robot, which represents the number of degrees of freedom of the robot motion, is defined:

$$\delta_m = \dim N[C_1^*(\beta_c)] = 3 - \text{rank}[C_1^*(\beta_c)].$$  \hfill (4.3.7)

Moreover, the degree of steerability $\delta_s$ represents the number of conventional centered orientable wheels of a mobile robot that can be oriented. It is formally defined as follows:

$$\delta_s = \text{rank}[C_{1c}(\beta_c)].$$  \hfill (4.3.8)

### 4.3.3 Classification

Following the aim of this chapter to generalize the properties of WMR and using the notation introduced regarding the degrees of mobility and steerability, it is possible to separate WMR in separate families. Since they depend on the aforementioned degrees, they are also used for families denomination as "$(\delta_m, \delta_s)$ mobile platform".

There are certain constraints on the degrees, which emerge due to practical issues. Firstly, the robot having more than one conventional fixed wheel should have them all on a single axle, while the centers of the conventional centered orientable wheels need to be outside this axle of the fixed wheels. Furthermore, the degree of mobility has upper limit 3, which is obvious, and lower bound 1, which is needed for a robot to be mobile.

$$1 \leq \delta_m \leq 3.$$  \hfill (4.3.9)

The degree of steerability can be in range:

$$0 \leq \delta_s \leq 2.$$  \hfill (4.3.10)

The lower bound holds for robots without centered orientable wheel, while the upper one corresponds to robots without fixed wheels.

The sum of both degrees, mobility and steerability, gives overall degrees of freedom that a robot can manipulate. This degree is in literature known as degree of maneuverability \cite{67}. Additional constraint in its context set the following inequality:

$$2 \leq \delta_m + \delta_s \leq 3.$$  \hfill (4.3.11)

The case when $\delta_m + \delta_s = 1$ is excluded, since than robot would only be able to drive around a fixed point, in a circle. Also, $\text{rank}[C_1^*(\beta_c)] \leq 2$, since otherwise degree of mobility would be equal to zero.

This leads to five different families of wheeled mobile robots, which are characterized hereafter.
(3,0) mobile platform \( (\delta_m = 3, \delta_s = 0) \)

These robots are called omnidirectional. The name originates from the fact that they are fully mobile in the plane, i.e., they are able to move at each instant in any direction (omni means all or every in Latin language). They have no conventional fixed or conventional centered orientable wheels \((N_f = 0, N_c = 0)\). An example of the omnidirectional platform with three omni wheels is given in Figure 4.9a.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{example.png}
\caption{Examples of mobile platforms}
\end{figure}

(2,0) mobile platform \( (\delta_m = 2, \delta_s = 0) \)

Robots belonging to this family have no conventional centered orientable wheels \((N_c = 0)\). They have either one or more conventional fixed wheel, only in the latter case, they need to be on a single common axle, to prevent rank of the matrix \(C_1\) to be bigger than 1. This restricts the mobility of the robot in terms of a trajectory \(\xi(t)\) such that the velocity \(\dot{\xi}(t)\) is constrained to belong to the two-dimensional distribution spanned by the vector fields \(R^T(\theta)v_1\) and \(R^T(\theta)v_2\), where \(v_1\) and \(v_2\) are two constant vectors spanning \(N[C_1^*(\beta_c)]\). One example of such a platform is presented in Figure 4.9b. It consists of two fixed wheels on the common axis and one additional off-centered orientable wheels.

(2,1) mobile platform \( (\delta_m = 2, \delta_s = 1) \)

No conventional fixed wheel \((N_f = 0)\) and at least one conventional centered orientable wheel \((N_c \geq 1)\) are forming this family of mobile platforms. An example with one center orientable wheel and two off-center orientable ones is given in Figure 4.10. In case there are multiple centered wheels, their orientations must be such that the degree of steer-ability stays equal to 1, i.e., \(\text{rank}[C_1(\beta_c)] = 1\). The velocity \(\dot{\xi}(t)\) is again constrained to belong to the two-dimensional distribution spanned by the vector fields, only now vectors \(v_1\) and \(v_2\) are a function of \(\beta_c\), which is of an arbitrary chosen conventional centered orientable wheel.

(1,1) mobile platform \( (\delta_m = 1, \delta_s = 1) \)

Mobile platform of these robots is configured such that it has one or more conventional fixed wheels with a single common axle. Additionally, they have one or more conventional centered orientable wheels, with the location away from the axle of the conventional fixed wheels (to prevent singularities) and their orientations coordinated
such that \( \text{rank}[C_1(\beta_c)] = 1 \) as well. The velocity \( \dot{\xi}(t) \) is constrained to belong to a one-dimensional distribution parameterized by the orientation angle of one arbitrarily chosen conventional centered orientable wheel. This type of mobile robots is called car-like robots (see Figure 4.11a), since they correspond to the configuration of a car model.

(1,2) mobile platform \((\delta_m = 1, \delta_s = 2)\)

These robots have no conventional fixed wheels \((N_f = 0)\) too, but at least two conventional centered orientable wheels \((N_c \geq 2)\). If there are more than two centered wheels, their orientations must be coordinated in such a way to keep the condition \( \text{rank}[C_1(\beta_c)] = \delta_s = 2 \) valid. The velocity \( \dot{\xi}(t) \) is constrained to belong to a one-dimensional distribution parameterized by the orientation angles of two arbitrarily chosen conventional centered orientable wheels of the robot. A configuration with two center orientable wheels and one off-center orientable wheels is presented in Figure 4.11b as an example.

4.4 Solvers for Kinematic Families

The solvers for kinematic families of Wheeled Mobile Robots derived in Section 4.3.3 can be expressed as parameterized geometric transformation between joint (wheel) space and Euclidean space. In terms of WMR, modeling the kinematics boils down to determining the motion of the robot from the geometry of the constraints imposed by the motion of the wheels. Kinematic analysis is based upon the assignment of coordinate axes within the robot and its environment, and the application of matrices to express the transformations between coordinate systems.
Generally, in stationary robotic manipulators, the relationship between velocities of the individual joints (wheels) and the velocities of the end-effector is expressed by the manipulator Jacobian matrix and it is called the forward kinematics. The opposite is also possible by the inverse of Jacobian matrix. On the other hand, when it comes to WMR, the so-called wheel Jacobian matrix links the velocities of each wheel on a WMR to the robot body velocities. The forward and inverse solutions are achieved by solving the kinematic equations-of-motion of all of the wheels at the same time, due to the fact that WMR are multiple dosed-link chains, i.e., they are mutually dependant [40]. In other words, the motion of one of the wheels directly influences the motion of the others, which is not the case in motion of stationary robotic manipulators.

At this point, the classification derived in the previous section will be used. The transformation matrices can be calculated by solving the constraints given in Equations (4.3.2) and (4.3.3). However, from an implementation point of view, there is additional classification that needs to be made. This is done with respect to the angle $\beta$, which corresponds to the orientation of the plane of the wheel with respect to CA (from, e.g., Figure 4.4). Angle $\beta$ can be either constant or time-varying. This is a crucial difference: In the former case, a transformation matrix needs to be calculated only once, offline and can be used afterwards online. In the latter case, the matrix needs to be dynamically updated at every instant depending on the value of the angle $\beta$. It is important to notice that constraints that are introduced by fixed or Swedish wheels are constant, due to the constant $\beta$, while orientable wheels bring in time-varying angle, i.e., time-varying constraints. This separates two groups of the platforms. The first group comprises of (3,0) and (2,0) mobile platforms with no off-center orientable wheels, and the second group contains all others, namely (2,1), (1,1) and (1,2) platforms. Families (3,0) and (2,0) which consist of castor wheels also belong to the second group, however they are rather rare.

For both groups the biggest issue is the fact that an inverse of one of the constraint matrices needs to be calculated. This becomes extremely problematic in case of over- or under-actuated WMRs. In that case, overdetermined or underdetermined system of linear equations needs to be solved. Since the system is given in the matrix form, the calculation of the generalized matrix inverse is required. General inverse is a nontrivial problem, since it requires complex numerical calculations. Therefore, it is elaborated hereafter.

### 4.4.1 Fixed Wheel Jacobian Matrix Calculation

As mentioned above, some wheel configurations allow the constraints on which the decoupling of kinematic models of WMRs depends to be fixed, i.e., they do not change over the time. An advantage of that fact can be taken and the transformation Jacobian matrix can be calculated beforehand. Then, having the Jacobian matrix, the decoupling is straightforward, and dynamic memory allocation and complex calculations are evaded.

Nowadays, this kind of platforms are very common, for instance, in the RoboCup Middle-Size League, like the TU/e TURTLE [3], the RFC robot [70], etc. Another nice example is the AMIGO robot [3], which is a part of the RoboEarth, RoboCup@Home and Bobbie projects [10]. It consists of four symmetrically arranged Swedish wheels, forming an over-actuated/sensored base (4 actuators/sensors), and thus requires solving an overdetermined system of equations (see Figure 4.12). Moreover, representative robots are Uranus [29] and Hilare [33].
For the implementation, another slight change needs to be introduced. Namely, the constraints given in Equations (4.3.2) and (4.3.3) can be merged, such that the system of linear equations is represented with the total matrices and not with separate ones for each constraint. The remark should be made here, that the rotation matrix $R(\theta)$ is left out, since the transformation is calculated in the robot frame, not in the world frame. By this, Equation (4.4.2) is achieved.

\[
\begin{bmatrix}
J_1 \\
C_1
\end{bmatrix} \dot{\xi} + 
\begin{bmatrix}
J_2 & 0 \\
0 & 0
\end{bmatrix}
\begin{bmatrix}
\dot{\psi} \\
0
\end{bmatrix} = 0.
\]

(4.4.1)

\[
JC_1 \dot{\xi} + JC_2 \begin{bmatrix}
\dot{\psi} \\
0
\end{bmatrix} = 0,
\]

(4.4.2)

Where $JC_1 = \begin{bmatrix} J_1 \\ C_1 \end{bmatrix}$ and $JC_2 = \begin{bmatrix} J_2 & 0 \\ 0 & 0 \end{bmatrix}$. Now, the wheel Jacobian matrix can be calculated by solving this system with respect to Euclidian coordinates $\dot{\xi}$.

\[
\dot{\xi} = -(JC_1)^\star \cdot JC_2 \begin{bmatrix}
\dot{\psi} \\
0
\end{bmatrix},
\]

(4.4.3)

where $(JC_1)^\star$ is a general matrix inverse. The reason for introduction of this general matrix inverse is the fact that the matrix $JC_1$ generally is non-square, i.e., it is generally not invertible. This issue has been treated thoroughly in literature (e.g., see [16], [8]). Numerous methods have been suggested, from one-sided inverse (left or right inverse), to Drazin inverse, Bott-Duffin inverse and finally Moore-Penrose pseudoinverse. The last one is the most-widely used and was first developed by Moore (see [37]). A common use of this method is to compute a least squares solution to a system of linear equations that lacks a solution (overdetermined system) or to find the minimum norm solution to a system of linear equations with multiple solutions (underdetermined system).
Moore-Penrose pseudoinverse

This pseudoinverse method guaranties the following properties for a matrix $M$ and its pseudoinverse $M^*$.

- $MM^*M = M$
- $M^*MM^* = M^*$
- $(MM^*)^H = MM^*$ (where symbol $H$ represents Hermitian transpose of a matrix)
- $(M^*M)^H = M^*M$

The pseudoinverse matrix can be constructed in several different ways. One solution is rank decomposition, but it is often source of numerical rounding errors \cite{16}, and hence, it is rarely used. Cholesky and QR decomposition overcome this problem. However, they are more suitable for analytical calculation, since they use recursion, which makes them difficult to implement. Finally, the Singular Value Decomposition (SVD) method is simple and accurate method and will therefore be used in this case.

The existence of the pseudoinverse using SVD can be proven as follows. A matrix $M$ of size $(n, m)$ can be decomposed into $M = USV^H$, where $U$ and $V$ are orthogonal square matrices containing the left and right singular vectors, and $S$ is an $(m, n)$ diagonal matrix, with as many nonzero entries in the diagonal as the rank of the matrix. The pseudoinverse is than given by:

$$M^* = VS^*U^H.$$ (4.4.4)

The pseudoinverse $S^*$ of a diagonal square matrix $S$ can be obtained just by taking the reciprocal of each non-zero element on the diagonal, leaving the zeros in place. In numerical computation, only elements larger than a small tolerance are taken to be nonzero, and the others are replaced by zeros.

It is easy to show now that $M^*$ is indeed pseudoinverse of $M$, i.e., that the first two properties mentioned above hold.

- $MM^*M = USV^HVS^*U^HUSV^H = USV^H = M$
- $M^*MM^* = VS^*U^HUSV^HVS^*U^H = VS^*U^H = M^*$

The uniqueness of the pseudoinverse matrix $M^*$ holds from the last two properties, but it depends on the size of the original matrix $M$. In case it has full column rank $(n > m)$, the following holds:

- $(MM^*)^H = \begin{bmatrix} I_m & O \\ O & O \end{bmatrix}^H_{n \times n} = \begin{bmatrix} I_m & O \\ O & O \end{bmatrix}_{n \times n} = MM^*$
- $(M^*M)^H = I^H_m = I_m = M^*M$

where $I_m$ represents the identity matrix of size $m$ and $O$ entries are zero matrices of the corresponding sizes. In case the original matrix $M$ has full row rank $(n < m)$, the following equalities hold:
4.4. Solvers for Kinematic Families

- \((MM^*)^H = I_n^H = I_n = MM^*
- \((M^*M)^H = \begin{bmatrix} I_n & O \\ O & O \end{bmatrix}_{m \times m}^H = \begin{bmatrix} I_n & O \\ O & O \end{bmatrix}_{m \times m} = M^*M

This concludes the proof.

For the implementation, a two-sided Jacobi SVD decomposition of a rectangular matrix is used \[42\]. For this purpose, a dependency to Eigen library is introduced \[26\]. Eigen is a C++ template library for linear algebra, which is used for handling matrices and vectors, and consists of numerical solvers and related algorithms. Appendix D offers the header file of the implemented component, together with the explanations.

4.4.2 Time-Varying Wheel Jacobian Matrix Calculation

Contrary to the previous, some of the wheel configurations of mobile platforms do not allow fixed constraints on which the decoupling of kinematic models of WMRs depends. Instead, the constraints are time-varying. A drawback of this is that the transformation Jacobian matrix cannot be calculated beforehand, but it needs to be dynamically updated at each instant. This hampers real-time behavior, although it is the only way that it can be implemented. Depending on the processor speed and sampling rate of the system, the calculation time can still be sufficient.

In particular, these platforms have, among others, at least one or more orientable wheels. Examples are HERO (Heathkit Educational ROBot) \[29\], Avatar \[6\] and Kludge \[30\] robots and probably the most known, PR2 robot \[46\] (see Figure 4.12), which has four off-centered orientable wheels and eight corresponding actuators/sensors (for orientation and rotation of the wheels).

For the sake of implementation, the original constraints \[4.3.2\] and \[4.3.3\] are merging together into one matrix. Remark should be made here, that the rotation matrix \(R(\theta)\) is left out for the same reason as in the previous section. By this, Equation \[4.4.6\] is achieved.

\[
\begin{bmatrix}
J_1(\beta_c, \beta_{oc}) \\
C_1(\beta_c, \beta_{oc})
\end{bmatrix}
\dot{\xi} + \begin{bmatrix} J_2 & 0 \\ 0 & C_2 \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\beta}_{oc} \end{bmatrix} = 0.
\]

\tag{4.4.5}

\[
JC_1(\beta_c, \beta_{oc}) \dot{\xi} + JC_2 \begin{bmatrix} \dot{\phi} \\ \dot{\beta}_{oc} \end{bmatrix} = 0,
\]

\tag{4.4.6}

where \(JC_1(\beta_c, \beta_{oc}) = \begin{bmatrix} J_1(\beta_c, \beta_{oc}) \\ C_1(\beta_c, \beta_{oc}) \end{bmatrix}\) and \(JC_2 = \begin{bmatrix} J_2 & 0 \\ 0 & C_2 \end{bmatrix}\). Now, the wheel Jacobian matrix can be calculated by solving this system with respect to Euclidian coordinates \(\dot{\xi}\).

\[
\dot{\xi} = -(JC_1)^*(\beta_c, \beta_{oc}) \cdot JC_2 \begin{bmatrix} \dot{\phi} \\ \dot{\beta}_{oc} \end{bmatrix}.
\]

\tag{4.4.7}

where \((JC_1)^*\) is again a general matrix inversion. The same approach (two-sided Jacobi SVD decomposition) for its calculation is used as in the previous section, so the explanation is omitted.
Performance assessment

In order to investigate whether this method meets its requirements, an analysis of the required time for calculation of the general matrix inverse is present here. An extensive set of tests has been conducted and the results are shown in Table 4.2 and Table 4.3. Depending on the number of wheels that constitute a mobile platform, the matrix size that needs to be inverted changes, i.e., number of rows increases with two for each additional wheel. Therefore, tests for inversion of rectangular matrices of different sizes were done and the performance measure is taken to be calculation time (given in milliseconds). Moreover, the times were recalculated into frequencies, in order to objectively present how usable this component is in a real-time application.

Table 4.2: Average and extremum required time for general matrix inverse with respect to the different matrix sizes

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Units</th>
<th>Average</th>
<th>Extremum</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4 × 3)</td>
<td>msec</td>
<td>0.448</td>
<td>0.666</td>
</tr>
<tr>
<td></td>
<td>kHz</td>
<td>22.9</td>
<td>15.0</td>
</tr>
<tr>
<td>(6 × 3)</td>
<td>msec</td>
<td>0.409</td>
<td>0.659</td>
</tr>
<tr>
<td></td>
<td>kHz</td>
<td>24.5</td>
<td>15.1</td>
</tr>
<tr>
<td>(8 × 3)</td>
<td>msec</td>
<td>0.490</td>
<td>0.607</td>
</tr>
<tr>
<td></td>
<td>kHz</td>
<td>21.1</td>
<td>14.5</td>
</tr>
</tbody>
</table>

For every matrix, ten calculation times are checked to obtain a realistic results. However, only their average and extremum are given in the tables. The full data tables, together with the specifications of the systems on which the tests were conducted, are presented in Appendix E. The extremum result is the "worst case", i.e., the case which takes the most time. As it can be noticed, increasing matrix size in average slows down the inverse calculation, but not significantly. Finally, it can be concluded that the calculation, in worst case, takes up to 0.666 milliseconds. In other words, it is sufficient as long as sampling frequency of application is below 15KHz, which is a rational assumption.

Table 4.3: Average and extremum required time for general matrix inverse with respect to the different matrix sizes

<table>
<thead>
<tr>
<th>Matrix size</th>
<th>Units</th>
<th>Average</th>
<th>Extremum</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10 × 3)</td>
<td>msec</td>
<td>0.513</td>
<td>0.633</td>
</tr>
<tr>
<td></td>
<td>kHz</td>
<td>19.8</td>
<td>15.0</td>
</tr>
<tr>
<td>(12 × 3)</td>
<td>msec</td>
<td>0.532</td>
<td>0.524</td>
</tr>
<tr>
<td></td>
<td>kHz</td>
<td>19.0</td>
<td>19.3</td>
</tr>
<tr>
<td>(14 × 3)</td>
<td>msec</td>
<td>0.524</td>
<td>0.654</td>
</tr>
<tr>
<td></td>
<td>kHz</td>
<td>19.3</td>
<td>15.2</td>
</tr>
</tbody>
</table>
Chapter 5

Conclusions and recommendations

5.1 Conclusions

In the past decades various tools have been developed in order to make an assortment of control functionalities possible. This holds for the functionalities that have to run in real-time, as well as for the control design tools. Nonetheless, these tools are generally expensive and solely oriented toward specific applications. Hence, the idea of the Systems and Control Library emerged and this project was initiated, with the following explicit problem statement:

*Develop and set a firm and clear software basis of an open source library which comprises all the functionalities and tools needed in the field of systems and control.*

The goal of defining a firm basis for the library in such a manner that it is compatible with and utilizable by different environments and platforms is achieved. During that process, a number of questions were treated and the decisions were thoroughly thought through. Moreover, by that, another subgoal was accomplished. All the decisions are well-reasoned and documented, which will enable future developers to contribute to the library with ease, as the most of the doubts they would potentially confront with have been covered in this report.

The importance of conducting the entire process of software development was recognized. Firstly, the motivation for the library was elaborated, which included a survey of currently available tools. The survey validated the idea and further emphasized the need of such a library. The conclusion was that none of the tools comprises all the requirements that are needed in the field of Systems and Control, in an open source format. The survey was followed by the analysis of the various aspects of the library, in terms of software. The library, as a congregation of code, needs to be compatible with other products and environments. The extensibility of the functionalities and thus code is a necessity. That demands a clear and firm software architecture. Reusability is recognized as the most important feature. The library is required to meet the needs of majority of its users, as it is intended to be utilized from various frameworks and for different functionalities. Specification of the required tools and functionalities in the
library accompanied. The list was motivated by possible applications and their needs.

Setting the library requirements brought about the software design phase, where the solutions to the raised demands were offered. 5Cs separation of concerns was chosen as software architecture approach. As an already proven concept in the design of complex software systems, it confirmed as a satisfying approach. This is reflected through the level of modularity and reusability of the software it offers. Hence, it enabled accomplishment of the determined requirements. Another short literature survey was carried out in order to motive the choice of programming language and its influence. C++ language was elected, due to the advantages object oriented programming enables. Additionally, the object oriented programming was complemented by component oriented programming, which imposed due to the nature of the application, i.e., control field.

Finally, issues of licensing, copyrights and documentation were processed. The well-documented software is a necessity and can determine the success of the library, since it makes the code appealing and easy to use.

The second part of the project included writing concrete tools, i.e., writing segments of the library. This process made clear existence of certain issues in the software design and led to its improvement. Two different topics were treated, namely digital filters and solvers for kinematic models of wheeled mobile robots. The first one is the core of the library and presents a natural starting point. The latter one was motivated by the need of the mobile platforms to be decoupled and controlled as sum of Single-Input, Single-Output (SISO) systems, in Euclidean space.

Possible digital filter realization methods were the matter of analysis, leading to the most suitable one for the real-time implementation. Numerical robustness of Second Transposed Direct-Form, in combination with advantages of high order filter decomposition into first and second order filters, shown the best results. Furthermore, available discretization methods were treated, which led to the conclusion that all of the methods have both, advantages and disadvantages, depending on the type of the application. Therefore, the decision was made that all of them should be implemented and the choice of method should be left to the user. Finally, corresponding discrete-time transfer functions were derived, allowing their implementation and thus ready-to-use digital filters.

On the other side, the required definitions regarding wheeled mobile robots (WMR) are given, starting from the robot posture and description of the wheels, to restrictions to robot mobility. Subsequently, the classification into five different classes of platforms with respect to degrees of mobility and steerability is introduced, which enables generalization of the WMR properties. However, from the implementation point of view, regrouping of those five families into two is preferable. The first group comprises platforms which have fixed transformation matrix between wheel and Euclidean space, while the second group consists of WMR with time-varying transformation matrix. This difference yields conceptually different implementations, since in the former case, the transformation matrix can be calculated offline, whereas in the latter case it needs to be recalculated online, depending on the time-varying parameters. The online calculation demands fast numerical algorithms, which were analyzed and the one that showed the best real-time behavior was implemented.
5.2 Recommendations

Along with these conclusions, a number of recommendations for further contributions can be stated.

- Various tools and functionalities need to be added. Some of them are listed in the Section 2.3.2 where the software requirements specification is elaborated, like transfer function and state space models, tools for optimization, Multiple-Input Multiple-Output (MIMO) control, rigid and flexible mode decoupling, etc. This does not conclude the list, but only suggests the essential tools that should be present. The list is indeed not limited and can be extended in case a certain application demands it.

- The 5Cs concept is a part which needs to be further enhanced. The theoretical part is clearly stated, however the full implementation still needs to be carried out. Mainly, this comprises the Directed Acyclic Graphs (DAG) data structure, which would enable Coordination concern in terms of ordering of components that need to work together and depend on each other’s results in a block scheme. It comes in useful also for composite control block schemes, where it should take care of the corresponding initialization, finalization and updating of the control diagram that it encapsulates and the components it consists of. This would make subsystem concept possible as well.

- In the context of the previous point, a tool for visualization of currently deployed library components can be designed. This would enable a clear and easy overview of the DAG data structure. Alternatively, already existing tools for visualization can be integrated.

- Bilinear transformation, as a type of continuous-time to discrete-time discretization, is used only as lowpass-to-lowpass type \( s \approx \frac{1 - z^{-1}}{1 + z^{-1}} \). However, lowpass-to-highpass \( s \approx \alpha \frac{1 + z^{-1}}{1 - z^{-1}} \) and parametric bilinear transformations \( s \approx \frac{1 - z^{-1}}{1 + rz^{-1}} \) are possible as well. Implementation of these transformations should be considered. For more information refer to [9] and [19].

- The implemented solvers for kinematic models of WMRs should be extensively tested on various types of platforms. Based on the tests, adaptations in the general matrix inversion algorithm can be made and other methods proposed in the report can be tried.

- The library should be published and made available online. For that purpose, a proper website or wiki page should be created. Additionally, this demands various examples and tutorials for the users and developers. Those examples should be well-detailed and comprehensive, but also clear and straightforward to understand.
Appendix A

Generic Formula for $s$-domain to $z$-domain Transformation

As explained in Chapter 3, discretization is an extremely important part of the control design process. This yields from the fact that the controllers are mainly designed in the continuous-time ($s$-domain) and then using discretization methods, translated into discrete-time, i.e., $z$-domain. The resulting form is possible to implement on the real-time system. These methods are, among others, the Euler backward and forward differentiation method, the bilinear (Tustin) transformation and Tustin transformation with prewarping.

All of these methods share the same property – they all transform continues-time transfer function $G(s)$ into discrete-time transfer function $G(z)$ by introducing transformation $s = f(z)$. This transformation can be also written as $s = \frac{p(z)}{q(z)}$.

The continuous-time transfer function can be represented as follows:

$$G(s) = \frac{B(s)}{A(s)} = \frac{b_0 + b_1 s + b_2 s^2 + \cdots + b_N s^N}{a_0 + a_1 s + a_2 s^2 + \cdots + a_M s^M},$$  \hspace{1cm} (A.0.1)

where $a_j$ and $b_i$ are feedback and feedforward coefficients, while $M$ and $N$ are orders of denominator and numerator, respectively. Inequality $N \leq M$ must hold, for filter to be feasible.

Now, if the transformation $s = \frac{p(z)}{q(z)}$ is introduced, the discrete-time transfer function
can be derived.

\[
G(s) = \frac{b_0 + b_1 \left( \frac{p(z)}{q(z)} \right) + b_2 \left( \frac{p(z)}{q(z)} \right)^2 + \cdots + b_N \left( \frac{p(z)}{q(z)} \right)^N}{a_0 + a_1 \left( \frac{p(z)}{q(z)} \right) + a_2 \left( \frac{p(z)}{q(z)} \right)^2 + \cdots + a_M \left( \frac{p(z)}{q(z)} \right)^M}
\]

\[
= \frac{b_0 q(z)^N + b_1 p(z) q(z)^{(N-1)} + \cdots + b_N p(z)^M q(z)^{(N-M)} }{a_0 q(z)^N + a_1 p(z) q(z)^{(N-1)} + \cdots + a_N p(z)^N}
\]

\[
= \sum_{i=0}^{M} b_{M-i} p(z)^{(M-i)} q(z)^{(N-M+i)}
\]

\[
\sum_{j=0}^{N} a_{N-j} p(z)^{(N-j)} q(z)^j
\]

\[
\text{(A.0.2)}
\]

Using the result of derivation in Equation [A.0.2], discretization methods listed above can be generalized and the transformation can be easily coded for any filter.

The similar approaches have been introduced in the literature (e.g., see [9], [19], [47]). However, they all aim at Pascal matrix formulation, which is not needed in this case, since the given formula can be directly implemented. For this purpose, a class with all required polynomial operations is designed and used. It is the constituent part of the library, with the name "Polynomial".
Appendix B

PID Anti-Windup

Despite the fact that many aspects of a control systems can be described using linear theory, some nonlinear effects must be accounted for in all controllers. Windup is such a phenomenon, which is a product of integral action and saturation interaction [59]. All real-time actuators essentially have limits: a motor has limited torque, a valve cannot be more than fully opened or fully closed, etc. For a control system operating in a wide range of conditions, it may happen that the control variable reaches the actuator limits. At that moment, the feedback loop is broken and the system runs as an open loop, i.e., the actuator will remain at its limit regardless of the process output [59]. If a controller with integrating action is used, the error will continue to be integrated. In other words, the integral term may become very large, that is, it "winds up". This requires the error to have an opposite sign for a long period before things return to normal and control variable gets out of the saturation. The implication is that any controller with integral action, e.g., PID, may give large transients when the actuator saturates. This is unwanted in real-time control and hence calls for a change. There are numerous techniques to account for this behavior and they are gathered under name Anti-windup.

Couple of the methods mostly used are explained here. However, only the last one, namely the back calculation method is implemented, since it yields best performance [59].

B.1 Set Point Limitation

The introduction of limiters on the set point variations is an attempt to avoid integrator windup. This prevents the controller output to reach the actuator limits. However, it frequently leads to conservative bounds and rather poor performance. Moreover, windup caused by disturbances is not accounted for using this method.

B.2 Conditional Integration

In this method, when the control is far from steady-state, the integration is turned off. In other words, integral action is only used when certain conditions are met, otherwise
the integral term is held constant. A couple of possible conditions that may cause the integral to be turned off exist.

The first one occurs when the control error is large. Another approach involves making the integrator constant when the actuator saturates. However, these concept have a disadvantage, because the controller may get stuck at a nonzero control error if the integral term has a large value at the time of switch off. Because of that, a better approach is to switch off integration when the controller is saturated and the integrator update is such that it causes the control signal to go even further into saturated [18]. For instance, if the controller becomes saturated at its upper bound, integration is then switched off if the control error is positive, but not if it is negative.

### B.3 Back Calculation Method

In the back calculation method, when the controller output saturates, the integral is recomputed so that its new value gives an output at the saturation limit. It is advantageous not to reset the integrator instantaneously but dynamically with a coefficient $k_{aw}$, since this would enable the system to give a fast response, but also not have too a large overshoot. Figure B.1 shows the block diagram for back calculation method where $e$ represent the error (difference between the reference and process output).

![Figure B.1: Anti integral windup using back calculation method](image)

The control system has an additional feedback loop that is generated by the difference between the output of the PID part $v$ and the complete controller output $u$, which forms an anti-windup error signal $e_s$. This error signal is then fed back to the integrator through gain $k_{aw}$. The signal equals zero when there is no saturation and thus will not effect the normal operation when the actuator does not saturate. On the other hand, when the actuator saturates, the signal $e_s$ is different from zero. The normal feedback path around the process is broken due to the process input, which remains constant. However, now there is a feedback path around the integrator. Because of this, the integrator output is driven towards a value such that the integrator input becomes zero. Gain $k_{aw}$ influences the performance greatly and can be tuned in a reasonable
manner\textsuperscript{35}. Generally, to prevent the integrator from saturating, $k_{aw}$ must be chosen small. Too small values, however decrease the controller performance\textsuperscript{60}. This method is chosen for the implementation, due to its proven performance features, especially in trajectory tracking\textsuperscript{45}. Using the implementation Transposed-Direct-Form II (TDF-II) structure, elaborated in Section 3.2, it is not possible to use the scheme in Figure 3.1 directly. The problem is the nonlinear term (saturation block). Therefore, an equivalent scheme needs to be proposed, which extracts the PID part and anti-windup part. A solution is presented in Figure B.2. Now, the PID controller can be implemented separately using TDF-II and then its output $v$ can be filtered by the anti-windup part, which would abrogate the integral action proportionally to the value by which the saturation limits are exceeded.

For the purpose of testing, this component was deployed through OROCOS\textsuperscript{©} and analyzed. The used simulation block scheme is given in Figure B.3. A simplest plant (basically just an integrator $\frac{1}{s}$) is taken to be controlled. For the reference, a step signal is chosen. The PID parameters are set ($k_p = 1$, $k_v = 1$, $k_i = 1$), as well as the saturation parameters ($-0.1 \leq u \leq 0.1$), such that only the influence of anti-windup part can be inspected. The results are shown in Figure B.4.

The upper subplot shows the output of the plant $y$, while the lower subplot depicts the input to the plant, i.e., control signal $u$. When there is no anti-windup action (the red curve), the output takes a long time to get to steady-state. This is directly caused by the windup of the integrator contained in the PID controller, which keeps integrating the servo error even if the input is saturated. On the other hand, when there is anti-windup action with coefficient $k_{aw} = 1$ (the blue curve), this problem is solved. The output converges to steady-state smoothly and relatively fast. Moreover, there are no longer strong output oscillations and the peaks of the control signal are flattened.
Figure B.4: PID Anti-windup test
Appendix C

Detailed Derivation of Standard Digital Filters’ Transfer Functions

This appendix offers detailed derivations of basic digital filters, using all discretization methods explained in the report. It relates to Chapter 3 of the report, where all the theory regarding the particular filters is given, while here only the explicit derivations are shown.

C.1 First Order Lowpass

The continuous-time transfer function is given by:

$$G(s) = \frac{1}{\frac{1}{2\pi f} s + 1}. \quad (C.1.1)$$

The discrete-time transfer function is derived using following methods.

- **Backward Differentiation Method**, \(G(s) \frac{s z^{-1}}{2\pi f} \rightarrow G(z)\)

  \[
  G(z) = \frac{1}{\frac{1}{2\pi f} z^{-1} + 1} = \frac{1}{\frac{1}{2\pi f T_s} - \frac{1}{2\pi f T_s} z^{-1} + 1} \quad (C.1.2)
  \]

- **Forward Differentiation Method**, \(G(s) \frac{s z^{-1}}{2\pi f} \rightarrow G(z)\)

  \[
  G(z) = \frac{1}{\frac{1}{2\pi f T_s} + 1} - \frac{1}{2\pi f T_s} z^{-1}.
  \]
\[ G(z) = \frac{1}{\frac{2\pi f}{T_s} + 1} \]
\[ = \frac{1}{\frac{1}{2\pi f T_s} z - \frac{1}{2\pi f T_s} + 1} \]  
\[ = \frac{1}{z^{-1}} \frac{1}{\frac{1}{2\pi f T_s} + (1 - \frac{1}{2\pi f T_s}) z^{-1}}. \]  

\[ \text{• Zero Order Hold (ZOH) Method, } G(z) \approx \frac{-1}{z} \mathcal{Z}\{1/G(s)\} \]
\[ G(z) = \frac{z-1}{z} \mathcal{Z}\{\frac{1}{s} \frac{1}{2\pi f s + 1}\} \]
\[ = \frac{z-1}{z} \mathcal{Z}\{\frac{1}{s} - \frac{1}{s + 2\pi f}\} \]
\[ = \frac{z-1}{z} \left(\frac{1}{z-1} - \frac{z}{z - e^{-2\pi f T_s}}\right) \]
\[ = 1 - \frac{z-1}{z - e^{-2\pi f T_s}} \]
\[ = \frac{z - e^{-2\pi f T_s} - z + 1}{z - e^{-2\pi f T_s}} \]
\[ = \frac{(1 - e^{-2\pi f T_s}) z^{-1}}{1 - e^{-2\pi f T_s} z^{-1}}. \]  

\[ \text{• Tustin’s Method, } G(s) \xrightarrow{s \approx \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}}} G(z) \]
\[ G(z) = \frac{1}{\frac{2}{2\pi f T_s} \frac{1-z^{-1}}{1+z^{-1}} + 1} \]
\[ = \frac{1}{\frac{1+z^{-1}}{2\pi f T_s} (1-z^{-1}) + 1 + z^{-1}} \]  
\[ = \frac{1+z^{-1}}{(\frac{1}{2\pi f T_s} + 1) + (1 - \frac{2}{2\pi f T_s}) z^{-1}}. \]  

\[ \text{• Tustin’s method with Prewarping, } G(s) \xrightarrow{s \approx \frac{1-z^{-1}}{1+z^{-1}}} G(z) \]

The decision on the choice of a prewarping frequency can be extremely complex. However, in this case, since we only have one critical frequency, the choice is straightforward and the cut-off frequency of the lowpass filter is chosen to be also the prewarping frequency.
\[ \omega_p = 2\pi f \quad \& \quad \alpha = \frac{\omega_p}{\tan (\omega_p T_s/2)}. \]  

84
\( G(z) = \frac{1}{\frac{1}{2\pi f}z^{-1} + 1} \)
\( = \frac{1 + z^{-1}}{\frac{1}{2\pi f}(1 - z^{-1}) + 1 + z^{-1}} \)
\( = \frac{1 + z^{-1}}{(\frac{a}{2\pi f} + 1) + (1 - \frac{a}{2\pi f})z^{-1}} \).
(C.1.7)

- **Matched Pole-Zero Method**, \( G(s) \xrightarrow{s_p = e^{-2\pi f T_s}, z_p = e^{\tau s}} G(z) \)

The continuous-time transfer function of this filter has no zeros and only one pole at \( s_p = -2\pi f \). This pole is mapped into discrete-time domain \( z_p = e^{-2\pi f T_s} \).

\( G(z) = \frac{1}{z - z_p} \)
\( = \frac{1}{z - e^{-2\pi f T_s}} \)
\( = \frac{1}{1 - e^{-2\pi f T_s}z^{-1}} \).
(C.1.8)

Additionally, we want to adjust the DC gain of \( G(z) \) so that it matches \( G(s) \).

\( G_s(0) = x \cdot G_z(1) \)
\( 1 = x \cdot \frac{1}{1 - e^{-2\pi f T_s}} \)
\( \Rightarrow x = 1 - e^{-2\pi f T_s} \).
(C.1.9)

After gain adjustment, \( G(z) \) becomes:

\( G(z) = \frac{(1 - e^{-2\pi f T_s})z^{-1}}{1 - e^{-2\pi f T_s}z^{-1}} \).
(C.1.10)

### C.2 Second Order Lowpass

The continuous-time transfer function is given by:

\( G(s) = \frac{1}{\frac{1}{(2\pi f)^2}s^2 + \frac{2D}{2\pi f}s + 1} \).
(C.2.1)

The discrete-time transfer function is derived using following methods.

- **Backward Differentiation Method**, \( G(s) \xrightarrow{s = \frac{z^{-1}}{T_s}} G(z) \)
\[ G(z) = \frac{1}{(2\pi f)^2 (\frac{z-1}{T_s})^2 + \frac{2D_p}{2\pi f} \frac{z-1}{T_s} + 1} \]

\[ = \frac{z^2 T_s^2}{(2\pi f)^2 (\frac{z^2 - 2z + 1}{T_s})^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

- Forward Differentiation Method, \( G(s) \underset{s \approx \frac{\Delta}{T_s}}{\longrightarrow} G(z) \)

\[ G(z) = \frac{1}{(2\pi f)^2 (\frac{z-1}{T_s})^2 + \frac{2D_p}{2\pi f} \frac{z-1}{T_s} + 1} \]

\[ = \frac{z^2 T_s^2}{(2\pi f)^2 (\frac{z^2 - 2z + 1}{T_s})^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

\[ = \frac{1}{(2\pi f)^2 (\frac{z}{T_s} - 1)^2 + \frac{2D_p T_s}{2\pi f} (\frac{z}{T_s} - 1) + T_s^2} \]

- Zero Order Hold (ZOH) Method, \( G(z) \approx \frac{z-1}{z} \mathcal{Z}\{\frac{1}{s} G(s)\} \)

The derivation and the final formula are left out from the report due to complexity.

- Tustin’s Method, \( G(s) \underset{s \approx 2 \frac{\Delta}{1+z^{-1}}}{\longrightarrow} G(z) \)

\[ G(z) = \frac{1}{(2\pi f)^2 (\frac{2}{1+z^{-1}})^2 + \frac{2D_p}{2\pi f} \frac{2}{1+z^{-1}} + 1} \]

\[ = \frac{(\frac{2}{1+z^{-1}})^2}{(2\pi f)^2 (1-z^{-1})^2 + \frac{2D_p}{2\pi f} \frac{2}{1+z^{-1}} (1 - z^{-1})(1 + z^{-1}) + (1 + z^{-1})^2} \]

\[ = \frac{1}{1 + 2z^{-1} + z^{-2}} \]

\[ = \frac{1}{1 + 2z^{-1} + z^{-2}} \]

\[ = \frac{1}{1 + 2z^{-1} + z^{-2}} \]

\[ = \frac{1}{1 + 2z^{-1} + z^{-2}} \]

- Tustin’s method with Prewarping, \( G(s) \underset{s \approx \frac{\Delta}{1+z^{-1}}}{\longrightarrow} G(z) \)

Even though this is a second order filter, it has only one critical frequency, cause it has two complex conjugate poles. Hence, the prewarping frequency is chosen
equal to the cut-off frequency.

\[ \omega_p = 2\pi f \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s / 2)}. \]  

\[ G(z) = \frac{1}{(2\pi f)^2 \left( (1 - z^{-1})^2 + \frac{2D_p}{2\pi f} \alpha (1 - z^{-1})(1 + z^{-1}) + (1 + z^{-1})^2 \right)} \]  

\[ G(z) = \frac{\alpha^2}{(2\pi f)^2} (1 - 2z^{-1} + z^{-2}) + \frac{2D_p}{2\pi f} \alpha (1 - z^{-2}) + 1 + 2z^{-1} + z^{-2} \]  

\[ G(z) = \frac{1}{2\pi f_p} \left( \frac{\alpha^2}{2\pi f_p} + 1 \right) + \left( 2 - \frac{\alpha^2}{(2\pi f)^2} \right) z^{-1} + \left( \frac{\alpha^2}{(2\pi f)^2} - \frac{2D_p \alpha}{2\pi f} + 1 \right) z^{-2}. \]  

- Matched Pole-Zero Method, \( G(s) \rightleftharpoons \frac{z_p = e^{sT_s}, z_{\gamma} = e^{sT_s}}{z_{p1}, z_{p2} \rightarrow G(z)} \)

First, the poles \( s_{p_{1,2}} \) of a continuous-time transfer function need to be computed. They are equal to the roots of a denominator:

\[ s_{p_{1,2}} = -\frac{2D_p}{2\pi f_p} \pm \sqrt{\left( \frac{2D_p}{2\pi f_p} \right)^2 - \frac{4}{(2\pi f_p)^2}} \]  

\[ s_{p_{1,2}} = -D_p \pm \frac{\sqrt{D_p^2 - 1}}{2\pi f_p}. \]  

Now, these poles are mapped into discrete-time domain \( z_{p1,2} = e^{\frac{-D_p \pm \sqrt{D_p^2 - 1}}{2\pi f_p} T_s} \).

\[ G(z) = \frac{1}{(z - z_{p1})(z - z_{p2})} \]  

\[ G(z) = \frac{1}{z^2 + (-z_{p1} - z_{p2})z + z_{p1}z_{p2}} \]  

\[ G(z) = \frac{1}{1 + (-z_{p1} - z_{p2})z^{-1} + z_{p1}z_{p2}z^{-2}}. \]  

Additionally, we want to adjust DC gain of \( G(z) \) so that it matches \( G(s) \).

\[ G_s(0) = x \cdot G_z(1) \]  

\[ 1 = x \cdot \frac{1}{1 - z_{p1} - z_{p2} + z_{p1}z_{p2}} \]  

\[ \Rightarrow \quad x = 1 - z_{p1} - z_{p2} + z_{p1}z_{p2}. \]  

After gain adjustment, \( G(z) \) becomes:

\[ G(z) = \frac{z^{-2}}{1 + (-z_{p1} - z_{p2})z^{-1} + z_{p1}z_{p2}z^{-2}}. \]
C.3 Weak Integrator

The continuous-time transfer function is given by:

$$G(s) = \frac{s + 2\pi f}{s}. \quad (C.3.1)$$

The discrete-time transfer function is derived using following methods.

- **Backward Differentiation Method**, $G(s) \xrightarrow{s \approx \frac{z-1}{T_s}} G(z)$
  $$G(z) = \frac{\frac{z-1}{T_s} + 2\pi f}{\frac{z-1}{zT_s}} = \frac{z - 1 + 2\pi f T_s z}{z - 1} = \frac{1 + (2\pi f T_s - 1) z^{-1}}{1 - z^{-1}}. \quad (C.3.2)$$

- **Forward Differentiation Method**, $G(s) \xrightarrow{s \approx \frac{z-1}{T_s}} G(z)$
  $$G(z) = \frac{\frac{z-1}{T_s} + 2\pi f}{\frac{z-1}{zT_s}} = \frac{z - 1 + 2\pi f T_s z}{z - 1} = \frac{1 + (2\pi f T_s - 1) z^{-1}}{1 - z^{-1}}. \quad (C.3.3)$$

- **Zero Order Hold (ZOH) Method**, $G(z) \approx \frac{z-1}{z} \mathcal{Z}\{\frac{1}{s} G(s)\}$
  $$G(z) = \frac{z - 1}{z} \mathcal{Z}\{\frac{1}{s} + \frac{2\pi f}{s^2}\} = \frac{z - 1}{z} \mathcal{Z}\{\frac{1}{s} + \frac{2\pi f}{s^2}\} = \frac{z - 1}{z} \left(\frac{z - 1}{z - 1} + \frac{2\pi f T_s z}{(z - 1)^2}\right) = 1 + \frac{2\pi f T_s}{z - 1} \frac{z - 1 + 2\pi f T_s}{1 - z^{-1}}. \quad (C.3.4)$$

- **Tustin’s Method**, $G(s) \xrightarrow{s \approx \frac{z-1}{z} \frac{1+z^{-1}}{1+z^{-1}}} G(z)$
\[ G(z) = \frac{2 \frac{1-z^{-1}}{T_s 1+z^{-1}} + 2\pi f}{\frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}}} \]
\[ = \frac{\frac{2}{T_s} (1 - z^{-1}) + 2\pi f (1 + z^{-1})}{\frac{2}{T_s} (1 - z^{-1})} \]
\[ = \frac{\frac{2}{T_s} + 2\pi f) + (2\pi f - \frac{2}{T_s}) z^{-1}}{\frac{2}{T_s} - \frac{2}{T_s} z^{-1}}. \quad \text{(C.3.5)} \]

- Tustin’s method with Prewarping, \( G(s) \xrightarrow{s \approx \alpha \frac{1-z^{-1}}{1+z^{-1}}} G(z) \)

The decision on the choice of a prewarping frequency is the same as for first order lowpass filter. Hence, the prewarping frequency is chosen equal to the cut-off frequency.

\[ \omega_p = 2\pi f \quad \& \quad \alpha = \frac{\omega_p}{\tan (\omega_p T_s / 2)}. \quad \text{(C.3.6)} \]

\[ G(z) = \frac{\frac{2}{T_s} + 2\pi f}{\alpha \frac{1-z^{-1}}{1+z^{-1}}} \]
\[ = \frac{\alpha (1 - z^{-1}) + 2\pi f (1 + z^{-1})}{\alpha (1 - z^{-1})} \]
\[ = \frac{(\alpha + 2\pi f) + (2\pi f - \alpha) z^{-1}}{\alpha - \alpha z^{-1}}. \quad \text{(C.3.7)} \]

- Matched Pole-Zero Method, \( G(s) \xrightarrow{z_p = e^{\frac{s}{T_s}}, z = e^{\frac{s}{T_s}}} G(z) \)

The continuous-time transfer function of this filter has only one zero at \( s_z = -2\pi f \) and only one pole at \( s_p = 0 \). These zero and pole are mapped into discrete-time domain \( z_z = e^{-2\pi f T_s} \) and \( z_p = e^0 = 1 \), respectively.

\[ G(z) = \frac{\frac{z - z_z}{z - z_p}}{\frac{1 - z_z z^{-1}}{1 - z_p z^{-1}}} \]
\[ = 1 - e^{-2\pi f T_s} z^{-1} \quad \text{(C.3.8)} \]

C.4 Lead Lag

The continuous-time transfer function is given by:

\[ G(s) = \frac{\frac{1}{2\pi f} s + 1}{\frac{1}{2\pi f} s + 1}. \quad \text{(C.4.1)} \]

The discrete-time transfer function is derived using following methods.
Appendix C

• Backward Differentiation Method, \( G(s) \xrightarrow{s \approx \frac{-1}{Ts}} G(z) \)

\[
G(z) = \frac{\frac{1}{2\pi f_c} + \frac{1}{z}}{1 + \frac{1}{2\pi f_p} - \frac{1}{z}} + 1
\]

\[
= \frac{\frac{1}{2\pi f_c} (z - 1) + z T_s}{1 + \frac{1}{2\pi f_p} (z - 1) + z T_s}
\]

\[
= \left( \frac{1}{2\pi f_c} + T_s \right) z - \frac{1}{2\pi f_c} + \left( \frac{1}{2\pi f_p} + T_s \right) z - \frac{1}{2\pi f_p}
\]

\[
= \left( \frac{1}{2\pi f_c} + T_s \right) - \frac{1}{2\pi f_c} z^{-1}
\]

\[
= \frac{1}{2\pi f_p} T_s - \frac{1}{2\pi f_p} \left( \frac{1}{2\pi f_c} + T_s \right) z^{-1}
\]

(C.4.2)

• Forward Differentiation Method, \( G(s) \xrightarrow{s \approx \frac{-1}{Ts}} G(z) \)

\[
G(z) = \frac{\frac{1}{2\pi f_c} + \frac{1}{z}}{1 + \frac{1}{2\pi f_p} - \frac{1}{z}} + 1
\]

\[
= \frac{\frac{1}{2\pi f_c} z - \frac{1}{2\pi f_c} + 1}{1 + \frac{1}{2\pi f_p} z - \frac{1}{2\pi f_p} + 1}
\]

\[
= \frac{1}{2\pi f_c} z + (1 - \frac{1}{2\pi f_c} T_s) z^{-1}
\]

\[
= \frac{1}{2\pi f_p} T_s + (1 - \frac{1}{2\pi f_p} T_s) z^{-1}
\]

(C.4.3)

• Zero Order Hold (ZOH) Method, \( G(z) \approx \frac{z - 1}{z} Z\left\{ \frac{1}{s} G(s) \right\} \)

\[
G(z) = \frac{z - 1}{z} Z\left\{ \frac{1}{2\pi f_c} \frac{1}{s} + \frac{1}{2\pi f_p} s + 1 \right\}
\]

\[
= \frac{z - 1}{z} Z\left\{ \frac{1}{s} + \frac{1}{2\pi f_c} \frac{1}{s} - \frac{1}{2\pi f_c} s + 1 \right\}
\]

\[
= \frac{z - 1}{z} Z\left\{ \frac{1}{s} + \frac{f_p}{f_c} - \frac{1}{s - 2\pi f_p} \right\}
\]

\[
= \frac{z - 1}{z} \left( \frac{z}{z - 1} + \frac{f_p}{f_c} - \frac{1}{z - e^{-2\pi f_p T_s}} \right)
\]

\[
= 1 + \left( \frac{f_p}{f_c} - 1 \right) \frac{z - 1}{z - e^{-2\pi f_p T_s}}
\]

\[
= \frac{z - e^{-2\pi f_p T_s} + \left( \frac{f_p}{f_c} - 1 \right) (z - 1)}{z - e^{-2\pi f_p T_s}}
\]

\[
= \frac{\frac{f_p}{f_c} + (1 - \frac{f_p}{f_c} - e^{-2\pi f_p T_s}) z^{-1}}{1 - e^{-2\pi f_p T_s} z^{-1}}
\]

(C.4.4)

• Tustin’s Method, \( G(s) \xrightarrow{s \approx \frac{z - 1}{z + 1}} G(z) \)

90
\[ G(z) = \frac{\frac{1}{2\pi f_0 T_s (1+z^{-1})} + 1}{\frac{1}{2\pi f_p T_s (1+z^{-1})} + 1} \]
\[ = \frac{\frac{1}{2\pi f_0 T_s (1-z^{-1})} + 1 + z^{-1}}{\frac{1}{2\pi f_p T_s (1-z^{-1})} + 1 + z^{-1}} \quad (C.4.5) \]
\[ = (1 + \frac{1}{2\pi f_0 T_s}) + (1 - \frac{1}{2\pi f_p T_s}) z^{-1} \]
\[ = (1 + \frac{1}{2\pi f_0 T_s}) + (1 - \frac{1}{2\pi f_p T_s}) z^{-1}. \]

- **Tustin’s method with Prewarping**, \( G(s) \xrightarrow{s \approx \alpha \frac{1-z^{-1}}{1+z^{-1}}} G(z) \)

In case of lead lag filter, choice of prewarping frequency is arguable. Basically, the whole region of frequencies between \( f_z \) and \( f_p \) is critical. However, since only one frequency can be matched using this method, it is chosen to be a frequency where the phase is maximal, \( f_c = \sqrt{f_z f_p} \). This decision relies on fact that in control theory, exactly this is the crucial frequency, where the most of the phase lead is needed and thus, accuracy is most important.

\[ \omega_p = 2\pi \sqrt{f_z f_p} \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)}, \quad (C.4.6) \]

\[ G(z) = \frac{\alpha}{2\pi f_z T_s (1+z^{-1})} + 1 \]
\[ = \frac{\alpha (1-z^{-1}) + 1 + z^{-1}}{2\pi f_p T_s (1-z^{-1}) + 1 + z^{-1}} \quad (C.4.7) \]
\[ = (1 + \frac{\alpha}{2\pi f_z T_s}) + (1 - \frac{\alpha}{2\pi f_p T_s}) z^{-1} \]
\[ = (1 + \frac{\alpha}{2\pi f_z T_s}) + (1 - \frac{\alpha}{2\pi f_p T_s}) z^{-1}. \]

- **Matched Pole-Zero Method**, \( G(s) \xrightarrow{z_p = e^{\omega_p T_s}, z_z = e^{\omega_p T_s}} G(z) \)

The continuous-time transfer function of a lead lag filter has only one zero at \( s_z = -2\pi f_z \) and only one pole at \( s_p = -2\pi f_p \). These zero and pole are mapped into discrete-time domain \( z_z = e^{-2\pi f_z T_s} \) and \( z_p = e^{-2\pi f_p T_s} \), respectively.

\[ G(z) = \frac{1 - z_z z^{-1}}{1 - z_p z^{-1}} \quad (C.4.8) \]
\[ = \frac{1 - e^{-2\pi f_z T_s} z_z^{-1}}{1 - e^{-2\pi f_p T_s} z_z^{-1}}. \]

Additionally, we want to adjust DC gain of \( G(z) \) so that it matches \( G(s) \).

\[ G_s(0) = x \cdot G_z(1) \]
\[ 1 = x \cdot \frac{1 - e^{-2\pi f_z T_s}}{1 - e^{-2\pi f_p T_s}} \]
\[ \Rightarrow x = \frac{1 - e^{-2\pi f_p T_s}}{1 - e^{-2\pi f_z T_s}}. \quad (C.4.9) \]
Appendix C

The continuous-time transfer function is given by:

\[
G(z) = \frac{1 - e^{-2\pi f_s T_s}}{1 - e^{-2\pi f_s T_s} e^{-2\pi f_s T_s}} = \frac{1 - e^{-2\pi f_s T_s}}{1 - e^{-2\pi f_s T_s} e^{-2\pi f_s T_s}} z^{-1}.
\]  

(C.4.10)

C.5 Skewed Notch

The continuous-time transfer function is given by:

\[
G(s) = \frac{1}{(2\pi f_p)^2 s^2 + \frac{2D_p}{2\pi f_p} s + 1}.
\]  

(C.5.1)

The discrete-time transfer function is derived using following methods.

- **Backward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{\zeta - 1}{T_s}} G(z) \)

\[
G(z) = \frac{1 - \frac{1}{(2\pi f_p)^2} (\frac{\zeta - 1}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta - 1}{T_s} + 1}{1 - \frac{1}{(2\pi f_p)^2} (\frac{\zeta - 1}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta - 1}{T_s} + 1} = \frac{1}{1 - \frac{1}{(2\pi f_p)^2} (\frac{\zeta - 1}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta - 1}{T_s} + 1} \frac{1}{(\frac{\zeta - 1}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta - 1}{T_s} + 1} \left(1 + 2\pi f_p T_s z^{-1} - \frac{2D_p}{2\pi f_p} T_s z^{-2}\right)
\]  

(C.5.2)

- **Forward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{\zeta}{T_s}} G(z) \)

\[
G(z) = \frac{1 - \frac{1}{(2\pi f_p)^2} (\frac{\zeta}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta}{T_s} + 1}{1 - \frac{1}{(2\pi f_p)^2} (\frac{\zeta}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta}{T_s} + 1} = \frac{1}{1 - \frac{1}{(2\pi f_p)^2} (\frac{\zeta}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta}{T_s} + 1} \frac{1}{(\frac{\zeta}{T_s})^2 + \frac{2D_p}{2\pi f_p} \frac{\zeta}{T_s} + 1} \left(1 + 2\pi f_p T_s z^{-1} - \frac{2D_p}{2\pi f_p} T_s z^{-2}\right)
\]  

(C.5.3)

- **Zero Order Hold (ZOH) Method**, \( G(z) \approx \frac{\zeta}{\frac{\zeta}{T_s} + 1} G(s) \)

The derivation and the final formula are left out from the report due to complexity.

- **Tustin’s Method**, \( G(s) \xrightarrow{s \approx \frac{\zeta}{T_s}} G(z) \)

92
\[ G(z) = \frac{1}{(2\pi f_p)^2} \left( \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}} \right)^2 + \frac{2D_p}{2\pi f_p} \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}} + 1 \]  
\[ \text{(C.5.4)} \]

\[ G(z) = \frac{1}{(2\pi f_p)^2} \left( \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}} \right)^2 + \frac{2D_p}{2\pi f_p} \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}} + 1 \]

\[ \text{(C.5.6)} \]

- Tustin’s method with Prewarping, \( G(s) \) \( \rightarrow \) \( G(z) \)

In case of a notch filter, the choice of prewarping frequency is a bit tricky and debatable. The whole region of frequencies between \( f_z \) and \( f_p \) is critical. However, since only one frequency can be matched using this method, it is chosen to be the frequency of the zeros (or poles in some other case, depending on the application). This decision becomes absolutely justified in the special case, i.e., a notch with \( f_z = f_p \), since exactly this, crucial frequency is chosen as a prewarping frequency.

\[ \omega_p = 2\pi \sqrt{f_z} \quad \text{or} \quad \omega_p = 2\pi \sqrt{f_p} \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)}. \]  
\[ \text{(C.5.5)} \]

\[ G(z) = \frac{1}{(2\pi f_p)^2} \alpha \left( \frac{1-z^{-1}}{1+z^{-1}} \right)^2 + \frac{2D_p}{2\pi f_p} \alpha \frac{1-z^{-1}}{1+z^{-1}} + 1 \]

\[ \text{(C.5.6)} \]
• Matched Pole-Zero Method, \( G(s) \xrightarrow{z_p = e^{s_p T_s}, z_z = e^{z_z T_s}} G(z) \)

First, the poles \( s_{p1,2} \) and zeros \( s_{z1,2} \) of a continuous-time transfer function need to be computed. They are equal to the roots of a denominator and numerator, respectively:

\[
s_{p1,2} = \frac{-2D_p}{2\pi f_p} \pm \sqrt{\frac{2D_p}{2\pi f_p}^2 - 4\frac{1}{(2\pi f_p)^2}} = -D_p \pm \sqrt{D_p^2 - 1} \tag{C.5.8}
\]

\[
s_{z1,2} = \frac{-2D_z}{2\pi f_s} \pm \sqrt{\frac{2D_z}{2\pi f_s}^2 - 4\frac{1}{(2\pi f_s)^2}} = -D_z \pm \sqrt{D_z^2 - 1} \tag{C.5.9}
\]

Now, these zeros and poles are mapped into discrete-time domain \( z_{z1,2} = e^{-\frac{D_z \pm \sqrt{D_z^2 - 1}}{2\pi f_s} T_s} \)

and \( z_{p1,2} = e^{-\frac{D_p \pm \sqrt{D_p^2 - 1}}{2\pi f_p} T_s} \), respectively.

\[
G(z) = \frac{(z - z_{z1})(z - z_{z2})}{(z - z_{p1})(z - z_{p2})} = \frac{z^2 + (z_{z1} - z_{z2})z + z_{z1}z_{z2}}{z^2 + (z_{p1} - z_{p2})z + z_{p1}z_{p2}} = \frac{1 + (-z_{z1} - z_{z2})z^{-1} + z_{z1}z_{z2}z^{-2}}{1 + (-z_{p1} - z_{p2})z^{-1} + z_{p1}z_{p2}z^{-2}} \tag{C.5.10}
\]

Additionally, we want to adjust DC gain of \( G(z) \) so it matches the same of \( G(s) \).

\[
G_s(0) = x \cdot G_z(1)
\]

\[
1 = x \cdot \frac{1 - z_{z1} - z_{z2} + z_{z1}z_{z2}}{1 - z_{p1} - z_{p2} + z_{p1}z_{p2}} \Rightarrow x = \frac{1 - z_{p1} - z_{p2} + z_{p1}z_{p2}}{1 - z_{z1} - z_{z2} + z_{z1}z_{z2}} \tag{C.5.11}
\]

After gain adjustment, \( G(z) \) becomes:

\[
G(z) = \frac{(1 - z_{p1} - z_{p2} + z_{p1}z_{p2})(1 + (-z_{z1} - z_{z2})z^{-1} + z_{z1}z_{z2}z^{-2})}{(1 - z_{z1} - z_{z2} + z_{z1}z_{z2})(1 + (-z_{p1} - z_{p2})z^{-1} + z_{p1}z_{p2}z^{-2})} \tag{C.5.12}
\]

### C.6 PD

The continuous-time transfer function is given by:

\[
G(s) = k_p + k_v s. \tag{C.6.1}
\]

The discrete-time transfer function is derived using following methods.
• Backward Differentiation Method, $G(s) \xrightarrow{s \approx \frac{z}{Ts}} G(z)$

$$G(z) = k_p + k_v \frac{z - 1}{zTs}$$

$$= \frac{(k_p Ts + k_v)z - k_v}{Ts}$$

$$= \frac{(k_p Ts + k_v) - k_v z^{-1}}{Ts}. \quad \text{(C.6.2)}$$

• Forward Differentiation Method, $G(s) \xrightarrow{s \approx \frac{z}{Ts}} G(z)$

Not possible to implement, due to no poles in the Continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.

• Zero Order Hold (ZOH) Method, $G(z) \approx \frac{z - 1}{z} \mathcal{Z}\{1 \frac{1}{s} G(s)\}$

$$G(z) = \frac{z - 1}{z} \mathcal{Z}\{\frac{1}{s}(k_p + k_v s)\}$$

$$= \frac{z - 1}{z} \mathcal{Z}\{k_v + \frac{k_p}{s}\}$$

$$= \frac{z - 1}{z} (k_v + \frac{k_p}{z} - 1)$$

$$= k_v \frac{z - 1}{z} + k_p$$

$$= k_v z - k_v + k_p$$

$$= \frac{z}{z}(k_v + k_p)z - k_v$$

$$= (k_v + k_p)z - k_v z^{-1}. \quad \text{(C.6.3)}$$

• Tustin’s Method, $G(s) \xrightarrow{s \approx \frac{2}{Ts} \frac{1 - z^{-1}}{1 + z^{-1}}} G(z)$

$$G(z) = k_p + k_v \frac{2}{Ts} \frac{1 - z^{-1}}{1 + z^{-1}}$$

$$= \frac{k_p (1 + z^{-1}) + k_v \frac{2}{Ts} (1 - z^{-1})}{1 + z^{-1}}$$

$$= \frac{(k_p + k_v \frac{2}{Ts}) + (k_p - k_v \frac{2}{Ts})z^{-1}}{1 + z^{-1}}. \quad \text{(C.6.4)}$$

• Tustin’s method with Prewarping, $G(s) \xrightarrow{s \approx \alpha \frac{2}{Ts} \frac{1 - z^{-1}}{1 + z^{-1}}} G(z)$

In case of a PD filter, choice of prewarping frequency is rather straightforward, since there is only one critical frequency.

$$\omega_p = 2\pi \frac{k_p}{k_v} \quad \& \quad \alpha = \frac{\omega_p}{\tan (\omega_p Ts/2)}. \quad \text{(C.6.5)}$$
Appendix C

\[ G(z) = k_p + k_v \frac{1 - z^{-1}}{1 + z^{-1}} \]
\[ = k_p (1 + z^{-1}) + k_v \alpha (1 - z^{-1}) \]
\[ = \frac{(k_p + k_v \alpha) + (k_p - k_v \alpha) z^{-1}}{1 + z^{-1}}. \]  \hspace{1cm} (C.6.6)

- **Matched Pole-Zero Method**, \( G(s) \xrightarrow{z_p = e^{s_p T_s}, z_v = e^{s_v T_s}} G(z) \)

   Not possible to implement, due to no poles in the Continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.

### C.7 Improved PD

The continuous-time transfer function is given by:

\[ G(s) = k_p + k_v \frac{N s}{s + N}. \] \hspace{1cm} (C.7.1)

The discrete-time transfer function is derived using following methods.

- **Improved Backward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z^{-1}}{T_s}} G(z) \)

\[ G(z) = k_p + k_v \frac{N \frac{z^{-1}}{T_s}}{\frac{z}{T_s} - 1} + N \]
\[ = k_p + k_v \frac{N z - N}{z - 1 + N z T_s} \]
\[ = \frac{(k_p + k_p N T_s + k_v N) z - k_p - k_v N}{(1 + N T_s) z - 1} \]
\[ = \frac{(k_p + k_p N T_s + k_v N) + (-k_p - k_v N) z^{-1}}{(1 + N T_s) - z^{-1}}. \] \hspace{1cm} (C.7.2)

- **Improved Forward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z^{-1}}{T_s}} G(z) \)

\[ G(z) = k_p + k_v \frac{N \frac{z^{-1}}{T_s}}{\frac{z}{T_s} - 1} + N \]
\[ = k_p + k_v \frac{N z - k_v N}{z - 1 + N T_s} \]
\[ = \frac{k_p z - k_p + k_p N T_s + k_v N z - k_d N}{z - 1 + N T_s} \]
\[ = \frac{(k_p + k_v N) z - k_p + k_p N T_s - k_d N}{z - 1 + N T_s} \]
\[ = \frac{(k_p + k_v N) + (-k_p + k_p N T_s - k_d N) z^{-1}}{1 + (-1 + N T_s) z^{-1}}. \] \hspace{1cm} (C.7.3)
- Improved Zero Order Hold (ZOH) Method, $G(z) \approx \frac{z^{-1}}{z} Z\left\{ \frac{1}{s} G(s) \right\}$

$$G(z) = \frac{z - 1}{z} \frac{1}{s} \left( k_p + k_v \frac{N}{s + N} \right) = \frac{z - 1}{z} \frac{1}{s} \left( k_p z^{-1} + k_v N \frac{z}{z - e^{-NT_x}} \right) = k_p + k_v N \frac{z - 1}{z - e^{-NT_x}}$$

$$\approx \frac{(k_p + k_v N) + (-k_p e^{-NT_x} - k_v N) z^{-1}}{1 - e^{-NT_x} z^{-1}} \quad \text{(C.7.4)}$$

- Improved Tustin’s Method, $G(s) \xrightarrow{s \approx \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}}} G(z)$

$$G(z) = k_p + k_v \frac{N \frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}}}{\frac{2}{T_s} \frac{1-z^{-1}}{1+z^{-1}} + N} = k_p + k_v \frac{N \frac{2}{T_s} - N \frac{2}{T_s} z^{-1}}{\frac{2}{T_s} - \frac{2}{T_s} z^{-1} + N + N z^{-1}} \quad \text{(C.7.5)}$$

$$= \frac{(k_p \frac{2}{T_s} + k_v N + k_v N \frac{2}{T_s}) + (-k_p \frac{2}{T_s} + k_v \frac{2}{T_s} N - k_v N \frac{2}{T_s} z^{-1})}{(\frac{2}{T_s} + N) + (-\frac{2}{T_s} + N) z^{-1}} \frac{z^{-1}}{1} \approx \frac{N}{1 - 1 z^{-1}}$$

- Improved Tustin’s method with Prewarping, $G(s) \xrightarrow{s \approx \alpha \frac{1-z^{-1}}{1+z^{-1}}} G(z)$

The choice of prewarping frequency is again straightforward, since there is only one critical frequency, the zero frequency (the pole frequency is determined by factor $N$, which should be considerably higher than zero frequency).

$$\omega_p = 2\pi \frac{k_p}{k_v} \quad \text{and} \quad \alpha = \frac{\omega_p}{\tan (\omega_p T_s/2)}. \quad \text{(C.7.6)}$$

$$G(z) = k_p + k_v \frac{N \alpha \frac{1-z^{-1}}{1+z^{-1}}}{\alpha \frac{1-z^{-1}}{1+z^{-1}} + N} = k_p + k_v \frac{N \alpha - N \alpha z^{-1}}{\alpha - \alpha z^{-1} + N + N z^{-1}} \quad \text{(C.7.7)}$$

$$= \frac{(k_p \alpha + k_v \alpha + k_v N \alpha) + (-k_p \alpha + k_v \alpha + k_v N \alpha) z^{-1}}{(\alpha + N) + (-\alpha + N) z^{-1}} \frac{z^{-1}}{1} \approx \frac{N}{1 - 1 z^{-1}}$$

- Matched Pole-Zero Method, $G(s) \xrightarrow{s_p = e^{\pi T_s}, z_z = e^{-\pi T_s}} G(z)$

First, the pole $s_p$ and zeros $s_z$ of a continuous-time transfer function are computed. They are equal to the roots of a denominator and numerator, respectively:

$$s_p = -N. \quad \text{(C.7.8)}$$
Now, these zeros and poles are mapped into discrete-time domain \( z = e^{-\frac{k_p N}{k_p + k_v N} T_s} \) and \( z_p = e^{-NT_s} \), respectively.

\[
G(z) = \frac{z - z_z}{z - z_p} = \frac{z - e^{-\frac{k_p N}{k_p + k_v N} T_s}}{z - e^{-NT_s}} = \frac{1 - e^{-\frac{k_p N}{k_p + k_v N} T_s} z z_z^{-1}}{1 - e^{-NT_s} z z_z^{-1}}.
\]

Additionally, we want to adjust DC gain of \( G(z) \) so it matches the same of \( G(s) \).

\[
G_s(0) = x \cdot G_z(1) \quad k_p = x \cdot 1 - e^{-\frac{k_p N}{k_p + k_v N} T_s} \quad \Rightarrow \quad x = k_p \frac{1 - e^{-NT_s}}{1 - e^{-\frac{k_p N}{k_p + k_v N} T_s}}.
\]

After gain adjustment, \( G(z) \) becomes:

\[
G(z) = k_p \frac{1 - e^{-NT_s} z}{1 - e^{-\frac{k_p N}{k_p + k_v N} T_s} z} \quad 1 - e^{-\frac{k_p N}{k_p + k_v N} T_s} z z_z^{-1}.
\]

### C.8 PID

The continuous-time transfer function is given by:

\[
G(s) = k_p + k_v s + k_i \frac{k_i}{s} = \frac{k_p s + k_v s^2 + k_i}{s}.
\]

The discrete-time transfer function is derived using following methods.

- **Backward Differentiation Method**, \( G(s) \xrightarrow{s \approx \frac{z}{T_s}} G(z) \)

\[
G(z) = k_p + k_v \frac{z - 1}{z T_s} + \frac{k_i}{z T_s} = k_p T_s (z^2 - z) + k_v (z^2 - 2z + 1) + k_i T_s^2 z^2 (z^2 - z) T_s^2 z^2 = (k_p T_s + k_v + k_i T_s^2) z^2 + (-k_p T_s - 2k_v) z + k_v T_s^2 - T_s z
\]

\[
= \frac{(k_p T_s + k_v + k_i T_s^2) z^2 + (-k_p T_s - 2k_v) z + k_v T_s^2 - T_s z}{T_s - T_s z^{-1}}.
\]
• Forward Differentiation Method, \( G(s) \rightarrow \frac{sT}{2T_s + 1} G(z) \)

Not possible to implement, due to a non-proper Continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.

• Zero Order Hold (ZOH) Method, \( G(z) \approx \frac{1}{z} \mathcal{Z}\{\frac{1}{s} G(s)\} \)

\[
G(z) = \frac{z-1}{z} \mathcal{Z}\{\frac{1}{s} (k_p + k_v s + \frac{k_i}{s})\} \\
= \frac{z-1}{z} \mathcal{Z}\{k_v + \frac{k_p + k_i}{s^2}\} \\
= \frac{z-1}{z} (k_v + k_p \frac{z}{z-1} + k_i T_s z) \\
= k_v \frac{z-1}{z} + k_v + k_i T_s \frac{z}{z-1} \\
= \frac{k_v}{z(z-1)}(z^2 - 2z + 1) + \frac{k_p(z^2 - z) + k_i T_s z}{z(z-1)} \\
= \frac{(k_v + k_p)z^2 + (k_i T_s - 2k_v - k_p)z + k_v}{z^2 - z} \\
= \frac{(k_v + k_p)z + (k_i T_s - 2k_v - k_p)z^{-1} + k_v z^{-2}}{1 - z^{-1}}.
\]

(C.8.3)

• Tustin’s Method, \( G(s) \rightarrow \frac{s}{2T_s + 1} G(z) \)

\[
G(z) = k_p + k_v \frac{2}{T_s} \frac{1 - z^{-1}}{1 + z^{-1}} + \frac{k_i}{T_s} \frac{1 - z^{-1}}{1 + z^{-1}} \\
= k_p + k_v \frac{2}{T_s} \frac{1 - z^{-1}}{1 + z^{-1}} + \frac{k_i}{T_s} \frac{1 - z^{-1}}{1 + z^{-1}} \\
= \frac{k_p}{T_s} (1 - z^{-2}) + \frac{k_v}{T_s} (1 - 2z^{-1} + z^{-2}) + \frac{k_i}{T_s} (1 + 2z^{-1} + z^{-2}) \\
= \frac{k_i}{T_s} (1 - z^{-1})(1 + z^{-1}) + \frac{2}{T_s} (1 - z^{-2}) + \frac{k_i}{T_s} + k_i - k_p \frac{2}{T_s} z^{-2} \\
= \frac{2}{T_s} - \frac{2}{T_s} z^{-2}.
\]

(C.8.4)

• Tustin’s method with Prewarping, \( G(s) \rightarrow \frac{s}{2T_s + 1} G(z) \)

A PID filter has a whole region of critical frequencies, i.e., between the two real zeros. Thus, it is very difficult to decide about the prewarping frequency. The best approach is to adapt prewarping frequency to the specific application. In this case, the prewarping frequency is chosen to be at the frequency of the first filter zero.

\[
\omega_p = 2\pi - k_v - \sqrt{k_v^2 - 4k_i k_p} \quad \& \quad \alpha = \frac{\omega_p}{\tan (\omega_p T_s / 2)}.
\]

(C.8.5)
Appendix C

\[ G(z) = k_p + k_v \alpha \frac{1 - z^{-1}}{1 + z^{-1}} + \frac{k_i}{\alpha(1 - z^{-1})} \]

\[ = k_p + k_v \alpha \frac{1 - z^{-1}}{1 + z^{-1}} + \frac{k_i(1 + z^{-1})}{\alpha(1 - z^{-1})} \]

\[ = k_p \alpha (1 - z^{-2}) + k_v \alpha (1 - 2z^{-1} + z^{-2}) + k_i(1 + 2z^{-1} + z^{-2}) \]

\[ \frac{1 + z^{-1}}{\alpha(1 - z^{-1})(1 + z^{-1})} \]

\[ = (k_p \alpha + k_v \alpha + k_i) + (2k_i - 2k_v \alpha)z^{-1} + (k_v \alpha + k_i - k_p \alpha)z^{-2} \]

\[ \alpha - \alpha z^{-2} \]

(C.8.6)

- Matched Pole-Zero Method, \( G(s) \overset{z_p=e^{sT_s}, z_z=e^{sT_z}}{\longrightarrow} G(z) \)

Not possible to implement, due to a non-proper Continuous-time transfer function, which makes the first coefficient of the discrete-time transfer function denominator zero.

C.9 Improved PID

The continuous-time transfer function is given by:

\[ G(s) = k_p + k_v \frac{Ns}{s + N} + k_i \frac{1}{s} \]

\[ = \frac{(k_p + k_v N)s^2 + (k_p N + k_i)s + k_i N}{s^2 + Ns} \]

(C.9.1)

The discrete-time transfer function is derived using following methods.

- Improved Backward Differentiation Method, \( G(s) \overset{s \approx \frac{z-1}{T_s}}{\longrightarrow} G(z) \)

\[ G(z) = \frac{(k_p + k_v N)(\frac{z-1}{T_s})^2 + (k_p N + k_i)(\frac{z-1}{T_s}) + k_i N}{(\frac{z-1}{T_s})^2 + N\frac{z-1}{T_s}} \]

\[ = (k_p + k_v N)(z^2 - 2z + 1) + (k_p N + k_i)T_s(z^2 - z) + k_i N T_s^2 z^2 \]

\[ z^2 - 2z + 1 + N T_s(z^2 - z) \]

\[ = (k_p + k_v N + k_p N T_s + k_i T_s + k_i N T_s^2) + \cdots \]

\[ (1 + N T_s) + (-2 - N T_s)z^{-1} + z^{-2} \]

\[ \cdots + (-2k_p - 2k_v N - k_i T_s - k_p N T_s)z^{-1} + \cdots \]

\[ (1 + N T_s) + (-2 - N T_s)z^{-1} + z^{-2} \]

\[ \cdots + (k_p + k_v N)z^{-2} \]

\[ (1 + N T_s) + (-2 - N T_s)z^{-1} + z^{-2} \]

(C.9.2)
• Improved Forward Differentiation Method, $G(s) \xrightarrow{s \approx \frac{1}{z}} G(z)$

\[
G(z) = \frac{(k_p + k_v N)(\frac{z}{T_s})^2 + (k_p N + k_i) \frac{z}{T_s} + k_i N}{(\frac{z}{T_s})^2 + N \frac{z}{T_s}}
\]

\[
= \frac{(k_p + k_v N)(z^2 - 2z + 1) + (k_p NT_s + k_i T_s)(z - 1) + k_i NT_s^2}{z^2 - 2z + 1 + NT_s(z - 1)}
\]

\[
= \frac{(k_p + k_v N) + \cdots}{1 + (-2 + NT_s)z^{-1} + (1 - NT_s)z^{-2}}
\]

\[
\begin{align*}
\cdots + (-2k_p - 2k_v N + k_p NT_s + k_i T_s)z^{-1} + \cdots \\
1 + (-2 + NT_s)z^{-1} + (1 - NT_s)z^{-2}
\end{align*}
\]

\[
\cdots + (k_p + k_v N - k_p NT_s - k_i T_s + k_i NT_s^2)z^{-2}
\]

\[
\begin{align*}
&= \frac{1}{1 + (-2 + NT_s)z^{-1} + (1 - NT_s)z^{-2}}
\end{align*}
\]

\begin{equation}
\text{(C.9.3)}
\end{equation}

• Zero Order Hold (ZOH) Method, $G(z) \approx \frac{z^{-1}}{z} \mathcal{Z}\{\frac{1}{s} G(s)\}$

\[
G(z) = \frac{z - 1}{z} \mathcal{Z}\{\frac{1}{s} (k_p + k_v \frac{N s}{s + N} + k_i \frac{1}{s})\}
\]

\[
= \frac{z - 1}{z} \mathcal{Z}\{k_p \frac{1}{s} + k_v \frac{N}{s + N} + k_i \frac{1}{s^2}\}
\]

\[
= \frac{z - 1}{z} (k_p \frac{z}{z - 1} + k_v N \frac{z}{z - e^{-NT_s}} + k_i T_s \frac{z}{(z - 1)^2})
\]

\[
= k_p + k_v N \frac{z - 1}{z - e^{-NT_s}} + k_i T_s \frac{1}{z - 1}
\]

\[
= k_p (z^2 + (-1 - e^{-NT_s})z + e^{NT_s}) + k_v N(z^2 - 2z + 1) + k_i T_s(z - e^{-NT_s})
\]

\[
= \frac{(z - e^{-NT_s})(z - 1)}{1 + (-1 - e^{-NT_s})z^{-1} + e^{-NT_s}z^{-2}}
\]

\[
\begin{align*}
\cdots + (k_p e^{NT_s} + k_v N - k_i T_s e^{-NT_s})z^{-2}
\end{align*}
\]

\[
\begin{align*}
&= \frac{1}{1 + (-1 - e^{-NT_s})z^{-1} + e^{-NT_s}z^{-2}}
\end{align*}
\]

\begin{equation}
\text{(C.9.4)}
\end{equation}

• Improved Tustin’s Method, $G(s) \xrightarrow{s \approx \frac{z-1}{z+1}} G(z)$

101
Appendix C

\[ G(z) = \frac{(k_p + k_v N)\left(\frac{\alpha}{1 + z^{-1}}\right)^2 + (k_p N + k_i)\frac{\alpha}{1 + z^{-1}} + k_i N}{(\frac{\alpha}{1 + z^{-1}})^2 + N\frac{\alpha}{1 + z^{-1}}} \]

\[ = \frac{(k_p + k_v N)\left(\frac{\alpha}{1 + z^{-1}}\right)^2(1 - 2z^{-1} + z^{-2}) + (k_p N + k_i)\frac{\alpha}{1 + z^{-1}}(1 - z^{-2}) - \cdots}{(\frac{\alpha}{1 + z^{-1}})^2(1 - 2z^{-1} + z^{-2}) + N\frac{\alpha}{1 + z^{-1}}(1 - z^{-2})} \]

\[ \cdots + k_i N(1 + 2z^{-1} + z^{-2}) \]

\[ = \frac{((k_p + k_v N)\left(\frac{\alpha}{1 + z^{-1}}\right)^2 + (k_p N + k_i)\frac{\alpha}{1 + z^{-1}} + k_i N)\cdots}{((\frac{\alpha}{1 + z^{-1}})^2 + N\frac{\alpha}{1 + z^{-1}} - 2((\frac{\alpha}{1 + z^{-1}})^2 - N\frac{\alpha}{1 + z^{-1}})z^{-2} \cdots + ((k_p + k_i N)(\frac{\alpha}{1 + z^{-1}})^2 + (k_p N - k_i)(\frac{\alpha}{1 + z^{-1}})^2 + k_i N)z^{-2}} \]

\[ \frac{((\frac{\alpha}{1 + z^{-1}})^2 + N\frac{\alpha}{1 + z^{-1}} - 2((\frac{\alpha}{1 + z^{-1}})^2 - N\frac{\alpha}{1 + z^{-1}})z^{-2} \cdots}{((\frac{\alpha}{1 + z^{-1}})^2 + N\frac{\alpha}{1 + z^{-1}} - 2((\frac{\alpha}{1 + z^{-1}})^2 - N\frac{\alpha}{1 + z^{-1}})z^{-2}}. \]

(C.9.5)

- Improved Tustin’s method with Prewarping, \( G(s) \xrightarrow{s \approx \omega_p \frac{1 - z^{-1}}{1 + z^{-1}}} G(z) \)

The same problem with the choice of a prewarping frequency exist here, since a PID filter has a whole region of critical frequency, which lays between two real zeros. The best approach is to adapt prewarping frequency to the specific application. In this case, the prewarping frequency is chosen to be at the frequency of the first filter zero.

\[ \omega_p = 2\pi \frac{-k_v - \sqrt{k_v^2 - 4k_i k_p}}{2k_p} \quad \& \quad \alpha = \frac{\omega_p}{\tan(\omega_p T_s/2)}. \quad \text{(C.9.6)} \]

\[ G(z) = \frac{(k_p + k_v N)\alpha\left(\frac{1}{1 + z^{-1}}\right)^2 + (k_p N + k_i)\alpha\frac{1}{1 + z^{-1}} + k_i N}{\alpha^2\left(1 - 2z^{-1} + z^{-2}\right) + N\alpha(1 - z^{-2}) \cdots + k_i N(1 + 2z^{-1} + z^{-2})} \]

\[ = \frac{(k_p + k_v N)\alpha^2(1 - 2z^{-1} + z^{-2}) + (k_p N + k_i)\alpha(1 - z^{-2}) + \cdots}{\alpha^2\left(1 - 2z^{-1} + z^{-2}\right) + N\alpha(1 - z^{-2})} \]

\[ = \frac{((k_p + k_v N)\alpha^2 + (k_p N + k_i)\alpha + k_i N)\cdots}{(\alpha^2 + N\alpha) - 2\alpha^2 z^{-1} + (\alpha^2 - N\alpha)z^{-2} \cdots + (-(2k_p - 2k_v N)\alpha^2 + 2k_i N)z^{-1} + \cdots} \]

\[ \cdots + ((k_p + k_i N)\alpha^2 + (-k_p N - k_i)\alpha + k_i N)z^{-2}} \]

\[ \frac{(\alpha^2 + N\alpha) - 2\alpha^2 z^{-1} + (\alpha^2 - N\alpha)z^{-2} \cdots + ((k_p + k_i N)\alpha^2 + (-k_p N - k_i)\alpha + k_i N)z^{-2}}{\alpha^2 + N\alpha - 2\alpha^2 z^{-1} + (\alpha^2 - N\alpha)z^{-2}.} \]

(C.9.7)

- Improved Matched Pole-Zero Method, \( G(s) \xrightarrow{z_p = e^{s_p T_s}, z_z = e^{z_z T_s}} G(z) \)

First, the poles \( s_{p1,2} \) and zeros \( s_{z1,2} \) of a Continuous-time transfer function need to be computed. They are equal to the roots of a denominator and numerator, respectively:

\[ s_{p1} = 0 \quad \& \quad s_{p2} = -N. \quad \text{(C.9.8)} \]
\[ s_{z1, 2} = \frac{-k_p N - k_i \pm \sqrt{(k_i + k_p N)^2 - 4k_i(k_p + k_v N)}}{2(k_p + k_i N)}. \]  

(C.9.9)

Now, these zeros and poles are mapped into discrete-time domain \[ z_{z1, 2} = e^{s_{z1, 2} T_s} \]
and \[ z_{p1} = 1, \quad z_{p2} = e^{-NT_s}, \] respectively.

\[ G(z) = \frac{(z - z_{z1})(z - z_{z2})}{(z - z_{p1})(z - z_{p2})} = \frac{z^2 + (-z_{z1} - z_{z2})z + z_{z1}z_{z2}}{z^2 + (-z_{p1} - z_{p2})z + z_{p1}z_{p2}} = \frac{1 + (-z_{z1} - z_{z2})z^{-1} + z_{z1}z_{z2}z^{-2}}{1 + (-z_{p1} - z_{p2})z^{-1} + z_{p1}z_{p2}z^{-2}}. \]

(C.9.10)
Appendix D

Examples of the Code

D.1 Digital Filters Example

The digital filters section of the library are gathered in the folder named "DFILTERS". It consists of all the mentioned filters and each of them has a corresponding header (".hpp") and source (".cpp") file. These are all, essentially, components. As the representative example of a digital filter, the code from the header file "DLeadLag.hpp" is given below.

First, basic information are given at the top of the file. This part consists of the file name, author name, date of the last edition and version of the file, and along with the rest of comments in the code, is used for generation of documentation. Secondly, copyright and licensing statements are declared. The rules presented in Section 2.6 of this report are reflected. Following this, the dependencies of the code using "#include" directive are defined. In this case, only a dependency to math, which is an integral part of C++ language, is introduced. Next, the required theory is laid out, together with the used references, to provide users and further developers with the information that was used for the design of the component.

Finally, the main code of the function starts with the definition of the class, which will represent a component, once it is instanced, i.e., an object is created. It typically requires the definition of a default constructor, copy-constructor, constructor and destructor. This is supplemented with the four main public functions, standard for each of the components (initialize, configure, update and finalize, see Figure 2.12). They reflect the software design concept presented in Section 2.4 of the report. The rest of public functions are pure getters, which enable users to obtain the computation results.

To sum up, this kind of component offers a user enough to directly deploy it for any possible application.

SCL/DFILTERS/DLeadLag.hpp

```cpp
1  /**
2   * @file DLeadLag.hpp
3   * @author Boris Mrkajic
4   * @date May, 2011
5   */
```
The DLeadLag class represents a digital lead-lag filter and it is a part of Systems & Control Library (section Digital Filters).

The name is rather straightforward, with the leading D as it represents digital implementation of a filter.

A general digital filter is characterized by its (discrete-time) transfer function, A lead-lag filter is on the other hand characterized by the two parameters, frequency of the zero and pole.

In order to obtain a discrete-time transfer function, an additional parameter is needed, which is sampling time. For more information see <http://en.wikipedia.org/wiki/Lead%E2%80%93lag_compensator>
For the implementation, the transposed-direct-form II (TDF-II) structure is chosen. An advantage of this structure is that the zeros effectively precede the poles in series order. Additionally, the transposition does not modify the transfer function of Single-Input, Single-Output (SISO) filter (see Oppenheim:1975).

Various discretization methods are available, namely Euler backward differentiation method, Euler forward differentiation method, standard Tustin method, Tustin method with prewarping (default), zero-order hold method and zero-pole matching method (see Haugen:2009, Ogata:1987 and Yang:2009). Method can be chosen at the moment of filter creation or through configuration function.

DLeadLag class uses the Polynomial class. The motivation yields from the way discretization is done, since most of the methods function such that they only introduce replacement of s (from s-domain) with a function of z (from z-domain), i.e. $s \rightarrow f(z)$. This makes it possible to generalize the routine and to make the derivation of discrete-time transfer function from the continuous-time transfer function automatic...

References:
@endverbatim

```java
class DLeadLag {
    public:
        // Order of a lead-lag filter by definition equals one (it ...
        // consists of one zero and one pole
        static const int filter_order = 1;

        // Default constructor
        DLeadLag();
        // Constructor that creates a filter with transfer function
        // that equals 1 (no filtering)
        /*
```
DLeadLag ( );

/// Copy constructor
/**
 * Constructor that copies a filter together with all its
 * features
 */
DLeadLag ( const DLeadLag & );

/// Constructor
/**
 * Constructor that creates a leadlag filter with the given
 * parameters
 * @param zero_freq frequency of the zero of the filter
 * @param pole_freq frequency of the pole of the filter
 * @param Ts sampling time used for filter discretization
 * @param method discretization method [ default = 1 ]
 * (1–Euler backward, 2–Euler forward, 3–Tustin,
 * 4–Prewarp Tustin, 5–Zero–order hold, 6–Zero–pole matching )
 */
DLeadLag ( double zero_freq, double pole_freq, double Ts, int ... method=1 );

/// Destructor
/**
 * Destructor that finalizes, i.e. resets parameters of a filter
 */
~DLeadLag ( );

/// Filter initialization
/**
 * Sets the parameters such that the transfer function equals 1
 * (no filtering) and output 0
 */
bool initialize ( );

/// Filter configuration
/**
 * Configures filter with the given parameters. Discretization
 * method may be left out (default method, Tustin with
 * prewarping, will be used)
 * @param zero_freq frequency of the zero of the filter
 * @param pole_freq frequency of the pole of the filter
 * @param Ts sampling time used for filter discretization
 * @param method discretization method [ default = 1 ]
 * (1–Euler backward, 2–Euler forward, 3–Tustin,
 * 4–Prewarp Tustin, 5–Zero–order hold, 6–Zero–pole matching )
 */
bool configure ( double zero_freq, double pole_freq, double Ts, ... int method=1 );

/// Filter update
/**
 * Function used for update of the current filter output,
 * depending on the given current input
 * @param input current filter input
 */
bool update ( double input );

/// Filter finalization
/**
 * Sets the parameters such that the transfer function equals 1
 * (no filtering) and output 0
 */
bool finalize ( );

/// Get numerator of the filter
/**
 * @return numerator of the filter as an array pointer
 */
double* getNumerator ( );

/// Get denominator of the filter
/**
 * @return denominator of the filter as an array pointer
 */

double* getDenominator();

/// Get previous inputs of the filter
/** @return previous inputs of the filter as an array pointer */
double* getPreviousInputs();

/// Get previous outputs of the filter
/** @return previous outputs of the filter as an array pointer */
double* getPreviousOutputs();

/// Get current output of the filter
/** @return current output of the filter */
double getOutput();

// Set the epsilon, value used to denote a number very close to zero (or some other specific value) within the bounds of double accuracy
bool setEpsilon(double epsilon);

private:

/// Denominator (coefficients) of a filter
double denominator[filter_order+1];

/// Numerator (coefficients) of a filter
double numerator[filter_order+1];

/// Previous inputs of a filter
double previous_inputs[filter_order];

/// Previous outputs of a filter
double previous_outputs[filter_order];

/// Output of a filter
double output;

double eps;

void savePreviousIO(double in, double out);

void cont2discrete(double* denDiscrete, double* numDiscrete, ... tDiscrete : Polynomial &p, Polynomial &q, Polynomial &denCont, ...)

void normalize(double aD discretized (z-domain) denominator, double bD discretized (z-domain) numerator);
D.2 Wheeled Mobile Platforms Decoupling Example

The solvers for kinematic models of wheeled mobile platforms are a section of the library which is gathered in the folder named "WMRDECOUPLING". It consists of the mentioned decoupling methods and again each of them has a corresponding header (".hpp") and source (".cpp") file. These are all, essentially, components. As the representative example, the code from the header file "WMRDecoupleFix.hpp" is given below.

The same structure as for the "DFILTERS" part of the library holds. First, basic information are given at the top of the file, followed by copyrights and licensing information. Next, the dependencies of the code using "#include" directive are defined. In this case, a dependency to math library, which is an integral part of C++ language, is introduced. Moreover, another essential dependency is introduced – dependency to the Eigen library. The reasoning is elaborated in Section 4.4. Furthermore, the required theory is laid out, together with the used references, to provide users and further developers with the information that was used for the design of the component.

Finally, the main code of the function starts with the definition of the class, which will represent a component, once it is instanced, i.e., an object is created. It typically requires the definition of a default constructor, copy-constructor, constructor and destructor. This is supplemented with the four main public functions, standard for each of the components (initialize, configure, update and finalize, see Figure 2.12). They reflect the software design concept presented in Section 2.4 of the report. The rest of public functions are pure getters, which enable users to obtain the computation results.

In summation, this kind of component offers a user enough to directly deploy it for any possible application.
Wheeled Mobile Robots (WMRs) decoupling with fix transformation

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer provided with the distribution.

THIS SOFTWARE IS PROVIDED BY TU/e "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL TU/e OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The views and conclusions contained in the software and documentation are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of TU/e.

The WMRDecoupleFix class represents a decoupling of a wheeled mobile robot with no orientable wheels (Swedish and fixed wheels only) from wheel to Euclidean space (frame) and it is a segment of Systems & Control Library (section WMRDECOUPLING).

A decoupling is executed for the purpose of Single-Input Single-Output (SISO) control of mobile robots equiped with wheels. Each WMR, i.e., wheeled mobile platform is configured with a number of wheels. All types of wheels have six parameters, namely three distances and three angles. In the case of this class, which offers solvers for platforms with only Swedish and fixed wheels, all the parameters are time-invariant. Therefore, the transformation matrix used for decoupling can be calculated only once, by rule at the moment of configuration of the platform (in particular, in the configure function of this class), and used later through update function, which needs to be provided by input. In this case, inputs are wheel velocities. For more information see Campion:2009.

For the matrix handling and numerical solving, in particular, calculation of the transformation matrix and dynamical calculation of the decoupled output in Euclidean frame, a dependency to the Eigen library is introduced. Eigen is a C++ template library for linear algebra, which is generally used for handling matrices and vectors, and consists of numerical solvers and related algorithms.

WMRDecoupleFix class uses the Wheel class. As already mentioned, all types of wheels have six parameters, which makes it possible to generalize them and for each wheel create an object with that parameter set.
typedef Eigen::Matrix<bool, Eigen::Dynamic, 1> VectorXb;

class WMRDecoupleFix
{

public:
  /// Default constructor
  /**
   Constructor that creates a wheel decoupling object with all
   the values set to zero
   */
  WMRDecoupleFix();

  /// Copy constructor
  /**
   Constructor that copies a wheel decoupling object, together
   with all its features
   */
  WMRDecoupleFix(const WMRDecoupleFix&);

  /// Constructor
  /**
   Constructor that creates a wheel decoupling object with the
   given parameters
   @param wheels_params parameters of the wheels (matrix of size ...
   (Nx6); N is a number of wheels, while each row has six ...
   parameters for each of the wheels)
   @param swedish_wheel vector of N boolean values (N is a number...
   of wheels) which are true if the wheel is of Swedish type...
   . Otherwise, it is set to false
   */
  WMRDecoupleFix(const Eigen::MatrixXd& wheels_params,
                 const ... VectorXb& swedish_wheel);

  /// Destructor
  /**
   Destructor that finalizes, i.e. resets parameters of a wheel
   decoupling object
   */
  ~WMRDecoupleFix();

  /// Wheel decoupling object initialization
  /**
   Function that sets all the parameters to zero
   */
  bool initialize(int _wheel_number);

  /// Wheel decoupling object configuration
  /**
   Function that sets parameters of the wheel decoupling object
   @param wheels_params parameters of the wheels (matrix of size ...
   (Nx6); N is a number of wheels, while each row has six ...
   parameters for each of the wheels)
   @param swedish_wheel vector of N boolean values (N is a number...
   of wheels) which are true if the wheel is of Swedish type...
   . Otherwise, it is set to false
   */
  bool configure(const Eigen::MatrixXd& wheels_params,
                 const ... VectorXb& swedish_wheel);

  /// Wheel decoupling object update

};
Function that calculates the Euclidean space coordinates from
the current velocities of the wheels
@param input vector of current velocities of the wheels
*
bool update(const Eigen::VectorXd & input);
/// Wheel decoupling object finalization
/**
  Function that resets all the parameters to zero
*/
bool finalize();
/// Get current decoupled output in Euclidean space
/** @return vector of size 3, with the current decoupled ...
  output in Euclidean space
*/
Eigen::Vector3d getOutput();
/// Get the transformation matrix from wheel to Euclidian ...
  space
/** @return transformation matrix from wheel to Euclidian ...
  space (matrix is of size (3x2*N))
*/
Eigen::MatrixXd getTransformationMatrix();

private:
/// Number of wheels
int wheel_number;
/// Wheels of the platform
Wheel* wheels;
/// Matrices of contracts along and orthogonal to the wheel ...
plain appended together
Eigen::MatrixXd JC1;
Eigen::MatrixXd JC2;
/// Transformation matrix of velocities from wheel to Euclidean...
  space
Eigen::MatrixXd wheel2Euclidean;
/// Decoupled output as a vector of doubles
Eigen::Vector3d output;
}
Appendix E

Test Results of the General Matrix Inverse Calculation

Here, a remark should be made. The tests were conducted on a machine equipped with Intel® Core™2 Duo CPU T9600, working at 2.8 GHz, and with 4 GB (DDR2 800 MHz RAM) of installed memory. Operating system was Ubuntu© 10.10 (Maverick Meerkat).

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Matrix size $\times 3$</th>
<th>Matrix size $\times 3$</th>
<th>Matrix size $\times 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(4 × 3)</td>
<td>(6 × 3)</td>
<td>(8 × 3)</td>
</tr>
<tr>
<td></td>
<td>msec       kHz</td>
<td>msec       kHz</td>
<td>msec       kHz</td>
</tr>
<tr>
<td>1</td>
<td>0.413       24.2</td>
<td>0.403       24.8</td>
<td>0.503       19.8</td>
</tr>
<tr>
<td>2</td>
<td>0.399       25.0</td>
<td>0.417       23.9</td>
<td>0.515       19.4</td>
</tr>
<tr>
<td>3</td>
<td>0.481       20.8</td>
<td>0.443       22.5</td>
<td>0.659       15.1</td>
</tr>
<tr>
<td>4</td>
<td>0.666       15.0</td>
<td>0.416       26.0</td>
<td>0.429       23.3</td>
</tr>
<tr>
<td>5</td>
<td>0.390       25.6</td>
<td>0.417       23.9</td>
<td>0.452       22.1</td>
</tr>
<tr>
<td>6</td>
<td>0.393       25.4</td>
<td>0.390       25.6</td>
<td>0.381       26.2</td>
</tr>
<tr>
<td>7</td>
<td>0.524       19.0</td>
<td>0.393       25.4</td>
<td>0.675       14.8</td>
</tr>
<tr>
<td>8</td>
<td>0.390       25.6</td>
<td>0.386       25.8</td>
<td>0.402       24.8</td>
</tr>
<tr>
<td>9</td>
<td>0.392       25.5</td>
<td>0.415       24.1</td>
<td>0.541       18.4</td>
</tr>
<tr>
<td>10</td>
<td>0.430       23.2</td>
<td>0.409       24.4</td>
<td>0.380       26.3</td>
</tr>
<tr>
<td>Average</td>
<td>0.448       22.9</td>
<td>0.409       24.5</td>
<td>0.490       21.1</td>
</tr>
<tr>
<td>Extremum</td>
<td>0.666       15.0</td>
<td>0.443       22.5</td>
<td>0.659       15.1</td>
</tr>
</tbody>
</table>
Table E.2: Required time for general matrix inverse with respect to the different matrix sizes

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Matrix size $(10 \times 3)$</th>
<th>Matrix size $(12 \times 3)$</th>
<th>Matrix size $(14 \times 3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>msec kHz</td>
<td>msec kHz</td>
<td>msec kHz</td>
</tr>
<tr>
<td>1</td>
<td>0.582 17.1</td>
<td>0.591 16.9</td>
<td>0.593 16.8</td>
</tr>
<tr>
<td>2</td>
<td>0.580 17.2</td>
<td>0.578 17.2</td>
<td>0.578 17.2</td>
</tr>
<tr>
<td>3</td>
<td>0.423 23.6</td>
<td>0.579 17.2</td>
<td>0.654 15.2</td>
</tr>
<tr>
<td>4</td>
<td>0.514 19.4</td>
<td>0.587 17.0</td>
<td>0.506 19.7</td>
</tr>
<tr>
<td>5</td>
<td>0.499 20.0</td>
<td>0.474 21.1</td>
<td>0.492 20.3</td>
</tr>
<tr>
<td>6</td>
<td>0.467 21.4</td>
<td>0.592 16.8</td>
<td>0.492 20.3</td>
</tr>
<tr>
<td>7</td>
<td>0.437 22.8</td>
<td>0.474 21.1</td>
<td>0.478 20.9</td>
</tr>
<tr>
<td>8</td>
<td>0.633 15.8</td>
<td>0.507 19.7</td>
<td>0.494 20.2</td>
</tr>
<tr>
<td>9</td>
<td>0.547 18.2</td>
<td>0.466 21.4</td>
<td>0.494 20.2</td>
</tr>
<tr>
<td>10</td>
<td>0.448 22.3</td>
<td>0.471 21.2</td>
<td>0.457 21.8</td>
</tr>
<tr>
<td>Average</td>
<td>0.513 19.8</td>
<td>0.532 19.0</td>
<td>0.524 19.3</td>
</tr>
<tr>
<td>Extremum</td>
<td>0.633 15.8</td>
<td>0.592 16.8</td>
<td>0.654 15.2</td>
</tr>
</tbody>
</table>
Bibliography


