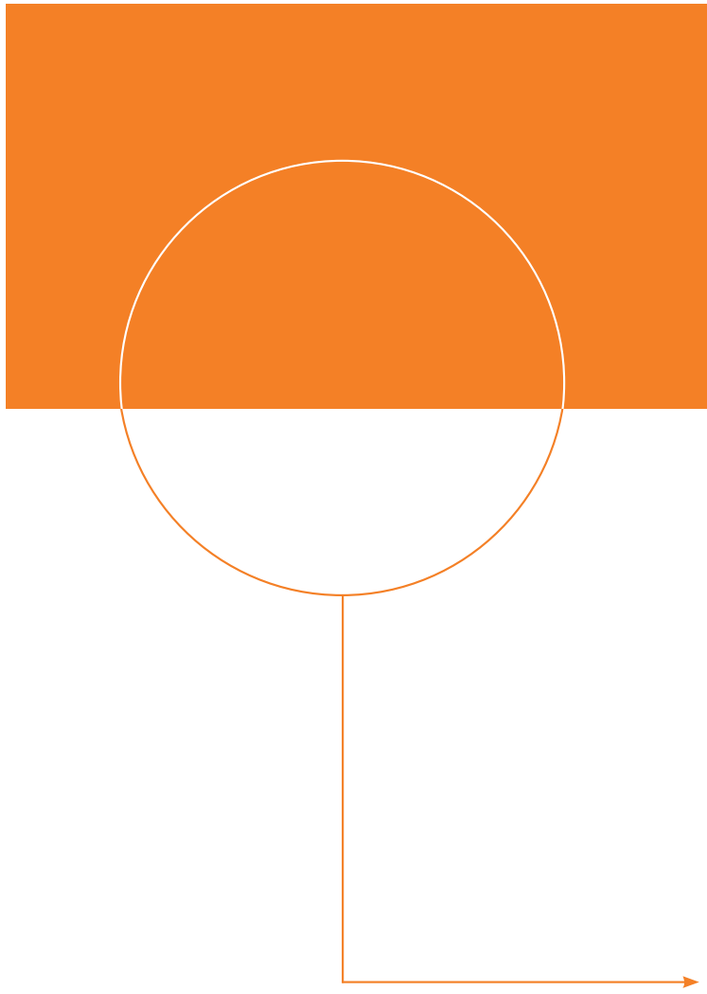


Master's thesis

2020

Are Fossli Viberg

**NTNU**
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics

Are Fossli Viberg

# Domain Knowledge in Real-Time Programming

June 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

# Domain Knowledge in Real-Time Programming

**Are Fossli Viberg**

Cybernetics and Robotics
Submission date: June 2020
Supervisor: Sverre Hendseth

Norwegian University of Science and Technology
Department of Engineering Cybernetics

# Preface

This master thesis concludes my 5-year master's program in engineering cybernetics at NTNU. This thesis is done in the 10th semester and constitutes half a semester of work, giving 15 study points. The work in its entirety is done in the period of the beginning of January to June 2020 which was of course a challenging period as the world faced the COVID-19 pandemic.

I would like to thank my supervisor Sverre Hendseth for all his help and assistance during this project and for many motivating, although sometimes confusing, stories and talks. A special thank goes to fellow students Aurora Haraldsen, which help was indispensable for refreshing my knowledge within control theory, and Øystein Molvik which has been a great conversation partner and friend throughout this semester.

Trondheim, 9.06.2020
Are Fossli Viberg

# Abstract

The use of domain knowledge in real-time programming is generally viewed as system-specific solutions that offers no generality for scheduling strategies. This thesis explored the term domain knowledge, what it is, where it can be used, and what it can offer toward real-time programming. The goal is to be able to break down the separation between application and real-time programming to offer alternatives ways of thinking of real-time.

A broad literature search was done covering cases of where and how the term real-time already has been broken down and implementation of domain knowledge is viable. This research has been summarized and categorized into four classes within real-time systems: control systems, WCET analysis, fail-safe systems, and dynamic deployment methods. Different suggestions on how to implement domain knowledge within these classes have been presented. A proof-of-concept example was carried out where an autonomous ship was modeled with the use of linear system theory. This example was modeled and simulated using Matlab, and its response analyzed using experience and concepts from control theory. Results from simulating this system showed that there is room for experimenting with different sampling frequencies while still experience acceptable behavior. This tells us that domain knowledge has the possibility to reduce the load on a CPU by reducing sampling frequency when the environment allows it.

In conclusion, there are a lot of viable methods for using domain knowledge in real-time programming. And although there are situations where how to obtain such information is unclear, many systems have well-defined domain knowledge available for more efficient real-time scheduling.

# Sammendrag

Bruk av domenekunnskap er generelt sett på som systemspesifikke løsninger som ikke kan tilby noe generalitet for strategier innen sanntidsprogrammering. Formålet med denne avhandlingen var å undersøke hva uttrykket "domenekunnskap" betyr, hvor det kan brukes og hva det har å tilby sanntidsprogrammering. Målet var å kunne bryte ned oppdelingen mellom applikasjon og programmering for å tilby nye måter å tenke på sanntid på.

Et bredt litteratursøk ble gjort for å dekke forskning gjort om hvordan sanntidsbegrepet har blitt brutt ned og implementering av domenekunnskap er en mulighet. Denne forskningen har blitt oppsummert og kategorisert i fore forskjellige klasser innenfor sanntidssystemer: reguleringssystemer, WCET analyse, fail-safe design og dynamiske distribusjonsstrategier. Forskjellige forslag ble gitt til hvordan man kan implementere domenekunnskap innenfor disse klassene. Et eksperiment ble utført hvor et autonomt styrt skip ble modellert ved bruk av lineær systemteori. Dette eksperimentet ble modellert og simulert med Matlab, og systemets respons analysert ved bruk av konsepter innen reguleringsteknikk. Responsen til systemet viste at det er rom for å eksperimentere med forskjellige tastefrekvenser og samtidig oppnå en akseptabel oppførsel. Dette forteller oss at domenekunnskap har muligheten til å redusere CPU-belastningen ved å redusere tastefrekvensen når domenet tillater det.

For å konkludere er det klart at det er mange egnede metoder for sanntidsprogrammering som benytter seg av domenekunnskap. Og selvom det finnes situasjoner hvor det er uklart hvordan man kan få tak i denne kunnskapen så finnes det mange systemer med godt definert domenekunnskap som kan brukes til mer effektiv sanntidsprogrammering.

# Table of Contents

# List of Acronyms

| | |
|---|---|
| **CPU** | Central Processing Unit |
| **QoS** | Quality-of-Service |
| **WCET** | Worst-Case Execution Time |
| **KF** | Kalman Filter |
| **IP** | Image Processing |
| **COTS** | Commercial off-the-shelf |

Chapter 1

# Introduction

## 1.1 Motivation

The definition of a real-time system is that the correctness of computation does not only depend on the output but also at which time the output is provided. This additional time constraint is what makes the field of real-time programming such a complex and interesting one as an output provided too late can cause serious consequences. For example are real-time computing systems crucial in flight-control, military weapon systems, and in chemical and nuclear plant control [13]. The strictest form of real-time is called hard real-time which does not allow a single time constraint to be violated. The concept of hard real-time was introduced already in 1973 [31] and has since formed a base for theory within the field of real-time programming. In the context of hard real-time, there is a decoupling between real-time programming and its application, where their domains do not need information from the other. in other words, hard real-time is a general theoretical approach independent on the domain it is applied in.

This decoupling, however, does not come without consequences. Let us say we have a system with a dedicated goal, and that smaller computational units within this program are called tasks. In hard real-time, these tasks are not allowed to miss their deadline of delivering their result, and their worst-case execution time (WCET) needs to be considered. In some systems, the WCET of a task might be a very rare and unusual outcome and using it as a reference can make for a very pessimistic system. Systems with a lot of dynamic variables such as control of autonomous vehicles often have tasks that are very time-varying and the WCET might not be a good measure of the actually needed constraint for correct behavior. To cope with these overly pessimistic real-time systems, high computational power must be given which often leads to a more powerful CPU than necessary. As technology only moves forward and grows more complex, the process of finding a WCET of a program is in itself a highly complex and non-linear exercise.

Today we can see that the use of smaller computers and embedded solutions are skyrocketing with the popularity of "smarter" everyday gadgets. Microcomputers are implemented all around us and we see a growing need for more power-efficient, but still

computational capable, CPUs. To make way for more complex microcomputers, while keeping the power consumption low, alternative solutions need to be explored.

## 1.2 Problem definition

In this thesis, the goal is to explore alternatives to classical hard real-time scheduling by making use of more system-oriented information. Using system-specific solutions has often been seen as a lost cause from the programming side as it does not offer any general design solutions or any re-usability. I believe this is too pessimistic and want to research how knowledge about the domain a system is working in can offer viable information towards more efficient scheduling for a real-time system. Hopefully, this would result in either being able to reduce the needed CPU power, reduce the usage of already dedicated CPU power or both. This kind of information will be called **domain knowledge** and will be further explained in the thesis.

A broad literature search will be done to research some areas within the field of real-time programming where the decoupling between programming and application has been broken. The idea behind this is to check out what has already been done in terms of alternative ways of real-time programming. This literature search will try to cover the main areas of interest within the real-time theory and will include WCET analysis, fail-safe real-time systems and dynamic and adaptive solutions. This will form a basis when an attempt to categorize domain knowledge into more general approaches is done later in the thesis. The categorization will also include suggestions on how to approach these classes using domain knowledge as an alternative to classical real-time programming.

As a proof-of-concept and by including control theory, the idea of using domain knowledge will be used in an example for a controller of an autonomous ship. This will be done by approaching the discrete updates of this controller as a real-time system where an update is viewed as a task. The theory needed to understand deductions and results of this example will be provided in its own chapter and will include both relevant concepts from real-time theory and control theory.

## 1.3 Outline

The report is structured as follows: First, a broad literature search is done to research areas where the use of domain knowledge may be relevant, or in some cases arguably already has been implemented. Following is a chapter with more textbook theory relevant to this thesis. Chapter 4 will go more in-depth on the information obtained in the literature review and further explore the (possible) use of domain knowledge in commonly used real-time programming methods. This chapter will also make try to categorize domain knowledge as a more general term rather than only system-specific. Chapter 5 is a more in-depth example of how domain knowledge can be utilized for an alternative way of scheduling a real-time system. Chapter 6 will show the results of a simulation in Matlab of the proof-of-concept and show how the system responds with different parameters. In the end, there will be a discussion that will take both the use of domain knowledge in general and the implemented example into consideration. A final conclusion will mark the end of this thesis.

Chapter 2

# Literature review

This chapter will review a wide selection of research done with regard to breaking the boundary of hard real-time. Optional ways of approaching real-time programming reaches out to every corner of the field with numerous different methods, and a full and well-covered overview would be a very demanding task. To apply some structure and make an overview possible to accomplish, three categories of research were chosen: WCET analysis, fail-safe design, and dynamic deployment methods.

## 2.1 WCET analysis methods

A real-time system depends on both output data and at what time the results are carried out. To ensure correct behavior, tasks in these systems have a deadline for when they need to be handled. Although the decision of what a deadline should be varied, it is very important to have a good estimate about the worst-case execution time, **WCET**, of a task. The most used method for WCET is a static off-line analysis where all possible time-delays are taken into consideration to ensure a safe estimation for the worst possible execution time for a task. As modern technology grows more and more complex even to find the worst possible path of execution for a task has turned into a big challenge. The growing complexity of programs also means that they require faster memory management, which makes way for use of dynamic memory allocation methods. To predict the execution time for these methods has shown to be quite the challenge as things like cache-misses can be very unpredictable. As WCET is especially important for crucial, hard real-time tasks, it is important that the estimate made is safe.

To ensure safe estimates the offline WCET analysis will very often be overly pessimistic[1] and require CPU power greatly overpowered for its purpose. To deliver the required CPU size is not a big problem for most systems as the technology in the subject have also improved massively over the years and will only affect already very big and complex

[1] Ensuring safe measurements is not the single reason for pessimism in static analysis and other reasons will be discussed later on in the thesis.

system. However, a large area of use for real-time systems are in embedded system, and here it is desirable to keep power consumption and memory use to a minimum, which is harder with a more powerful CPU than necessary.

Alternative ways to do WCET analysis has been a hot topic for many years as the pessimism of static offline analysis has been a problem for a long time. In this chapter, alternative approaches will be presented where more system specific information is utilized to ensure tighter WCET estimates.

### 2.1.1 Introduction to measurement-based WCET analysis

The pessimism of static WCET analysis is a commonly accepted problem within the field of real-time systems. This has led to a lot of research for alternative and more efficient WCET analysis methods and this has been a highly active field for some years now. The work presented in this section will describe approaches where timing measurements are utilized for alternative ways of analyzing WCET. These measurements are done in run-time utilizing hardware clocks and other built-in methods that is typical for smaller systems structured around a CPU.

One approach is by what is called end-to-end measurements. This is done by measure the time from execution start to execution is finished of a program. With very simple programs it might be sufficient to do an end-to-end timing measurement to analyze the WCET, but this is in general seen as a non-safe approach as it cannot guarantee a capture of the worst-case execution for the program. When end-to-end measurements is done with exhaustive testing and carefully put safety margin, they can bring useful results. A statistical approach can be done where the results can form a probability distribution of what the WCET of the program is. However, for smaller programs, the effort of doing enough tests to get a good probability function might be too big for its purpose. The safety margins might also bring too much slack on the WCET estimate making the results too pessimistic and therefore bring no real solution to the original problem [6]. End-to-end measurements might not be the solution for replacing static WCET analysis, but as this section will show there are other solutions suggested which are both safer and more efficient.

### 2.1.2 Hybrid approaches and automatic test data generation

The growing complexity of control systems is also causing problems for the established static WCET analysis methods [40]. A hybrid approach is suggested where both the well-established static analysis methods and time measurements in run-time on real hardware are utilized to estimate WCET of a program. The approach described involves some predesigned steps:

1. **Static Analysis:** The program to run, in this case, written in C source code, is analyzed and the structure is derived. It is mentioned that by focusing on C code rather than more common methods on object-code level, this analysis will give a high level of portability as the C language is well-established in control systems.

2. **Partitioning:** In this part the code is automatically divided into smaller segments, where each segment offers a lower level of complexity to the code.

3. **Test data generation:** This step has three levels. The first is to apply randomly generated input data to cover many possible paths in the program. The second is to apply an evolutionary algorithm to seek out paths that are harder to find. The final level, to find the very last possible paths, is to use model checking for test-data generation.

4. **Execution time measurements:** The test-data from the last step is used to execute found paths in the program segments and automatically generated code instrumentation creates timing information.

5. **Calculation step:** The created timing information is combined to a total WCET by using structural information from the first step.

Many articles have been written about this approach ([26] [41]) and seem to agree on this kind of framework. It is important to note that there is a lot more to this approach to be able to understand the full extent of it. Subjects like automatic test data generation, evolutionary algorithms, and model checking play a vital role to realize this kind of WCET estimate.

### 2.1.3 Safe measurement-based WCET estimation

One of the reasons static WCET analysis is such widely used is that it can guarantee safe[2] estimations, which is not necessarily true with a measurement-based approach. Deverge and Puaut have suggested an approach in their article [16] where they use hardware measurements during run-time of a program to estimate WCET. Structural testing methods are used to generate input test-data and use these to measure execution time for program paths. The testing method used is called program clustering [45] and this process divides the path into smaller, less complex, segments of code. The clustering suggested by Deverge and Puaut is to apply automated test data generation in an iterative way on smaller and smaller parts of the program until segments are small enough for an exhaustive path enumeration inside it to be tractable. Program measurement methods are exploited to fetch execution time from these segments.

For the measurements to be considered safe, some points of unpredictability are made which need to be handled.

1. **Global mechanisms.** E.g. cache, virtual memory translation, and branch prediction.

2. **Variable latency instructions.** E.g. integer multiplication instruction varies with the length of the integer.

3. **Statistic execution interference.** E.g. unpredictability due to DRAM refresh.

If these points are addressed and accounted for measurements can be used for safe WCET estimates.

[2] "safe" in this context means that the estimated WCET is guaranteed to be greater or equal to the actual WCET.

### 2.1.4 Utilizing online WCET data

The benefits of online WCET estimates over offline estimation is generally a popular point of discussion. One suggestion assumes that a WCET can be calculated by an estimation algorithm at the release of a task [23]. The article discusses the possible existence of such an estimation algorithm, which is not obvious in itself, but that it might be a possibility. A deployed program in a real-time setting will always terminate in bounded time, assuming correct behavior from the system, which is a case where an estimation algorithm is applicable. These algorithms could also be approached by using a priori information[3] together with automatically generated test data, similar to the previously mentioned clustering method. By fetching tighter online WCET estimations some new potential improvements occur for a system such as:

- Overall tighter WCET will make for more efficient scheduling.
- Improved algorithms for reclaiming the CPU capacity allocated to a task but not used.
- Programs could offer more modes of operations.
- Improved algorithms for dynamic deployment of tasks.

In the thesis written by Bakkevig [3], this idea has been utilized to suggest an interesting approach using different variants of the same task. Bakkevig suggests that the online analysis upon release of a task is done using some release-specific task parameters like input and program state, which can give a trustworthy and yet tighter WCET than an offline estimate. Each task will have different variants of itself, where there is one preferred variant and some optional ones which are less complex and maybe more inaccurate, yet acceptable. When a task is added to the scheduler, an online WCET estimator is used for the highest priority variant of that task. The scheduler checks if all deadlines can be met, and if not, a degraded variant of the task is checked with the same analysis. The first task variant to fulfill the acceptance criteria is added to the system, if no variants are accepted, the task is aborted.

The approach uses the priority of the main versions of each task, but priority can also be assigned freely on the degraded variants. This is because there may arise situations where it is more desirable to degrade a higher priority task to lower variant rather than aborting a lower prioritized task.

### 2.1.5 Probabilistic WCET analysis

Probabilistic analysis methods are statistical approaches that combine measurements and analytical tools to create estimates of a the execution time for a program, a specific method like this can be found in [6]. The approach is so-called *grey box*, where the smaller parts of the program are abstracted away and measured on input and output. These smaller parts are called execution profiles and are mostly used in the context of execution time in this approach and called execution time profiles, ETP for short. The granularity of these ETPs

[3] A priori - knowledge, values, and arguments known before execution

are defined from the first instruction of a unit in a pipeline till the first unit of the next unit into the pipeline. To obtain such ETPs from a program code three different approaches are suggested:

- real measurements on a processor
- by using a simulator
- analytical methods

Where all of them are useful in different ways but also provokes some problems and points of inaccuracy.

The idea of this approach is to divide program code into ETPs and obtain their distribution function and with that their WCET. The final WCET for the program will be obtained by combining these ETPs which is done based on their dependencies. The dependencies between ETPs is the main source for difficulties in this approach, and is categorized into three different scenarios:

1. all ETPs are independent
2. the dependencies between the ETPs are known
3. the information about the ETP dependencies are unknown

The first scenario can combine the ETPs by using convolution and statistics, and the article provides a thorough description of the algebraic approach. The second scenario makes use of correlations, that are assumed known, and similar statistics to combine the ETPs. But these first two scenarios can at best simulate good approximations of real measures as it is highly probable that the dependencies are not fully known. Therefore, in the third scenario, the article describes an approach where the "worst" correlation between two ETPs must be chosen. The worst in this context means the strongest positive correlation between two ETPs and by choosing this one, one can guarantee to not underestimate the combination. the article provides an algorithm to choose the worst correlation.

An explicit tool for this approach called the pWCET, p = probabilistic, is described in [7]. This tool structures a program into a syntax tree, with the basic units as leaf nodes. After the syntax tree is generated a trace generation process produces execution traces to capture the execution time of basic blocks. After the traces are made, the tool produces a distribution function to each of them to capture their execution time. A timing program is generated to traverse the tree and generates the WCET path which is calculated afterward. In the end, the results are analyzed and graphically shown to the user.

### 2.1.6 Automatic Timing Model Generation by CFG Partitioning and Model Checking

A method for partitioning code int measurable segments is suggested in [42] and is called CFG partitioning. CFG, control flow graph, is a graphical representation of code which divides the code into basic units called *basic blocks*. A basic block is such that the control flow of the program enters and leaves the block without the possibility to branch out to

further paths, except into another basic block. The CFG is therefore a tree-structured graph with linked basic blocks. Another important aspect of CFG is the program segment, PS, which is a subgraph of the CFG with only a single entry.

The test data generation is done in two steps: first by using heuristic methods, for example by genetic algorithms, up until a coverage constraint for paths in the code is reached, and then the rest of the paths are found by traditional model checking. The use of heuristic methods is because it is considerably more cost-efficient for computational power, which is a big issue with the use of model checking. To further cut computational cost several other optimization approaches can be used to analyze the code. Examples:

- Line-variable analysis
- Statement concatenation
- Variable range analysis
- Variable initialization
- Dead variable and code elimination
- Sal model checker [15]

## 2.2 Fail-Safe real-time systems

A **fail-safe** system design has built-in solutions that can take the system to a safe state should there occur a failure or an error. It can be compared to a safety net for trapeze artists, should the artist miss a jump he or she would land in the net, the fail-safe state. When a system enters the safe state it should minimize, or ideally remove, any possible damage to people, environment, or other parts of the system. This is a very useful system design for real-time systems with critical tasks where failures and errors can cause serious consequences. For a good fail-safe design, it is critical with high error detection coverage in the system, which can be provided with watchdogs, heartbeat signals, and periodic status checks. A related term to fail-safe is called **fail-operational** design. A fail-operational system will provide a minimum level of service after the occurrence of a fault and even in the case where there is no fail-safe state to be reached. An example of how important the fail-safe state can be is with the tragic incident in the Chernobyl nuclear power plant where a poorly implemented fail-safe is believed to have caused the final explosion instead of preventing it [34].

The use of fail-safe design in real-time systems is a very interesting topic that provides a lot of research. The rest of this section will cover some of the research done in this field.

### 2.2.1 Fail-Awareness

A common term within the field of fail-safe programming is **fail-awareness**[19], which is an approach to construct fail-safe applications. This is based on systems that make us of standardized programs and hardware, so-called commercial-off-the-shelf (COTS) products, to create systems with hard real-time properties. The problem with the use of COTS

products is that there is no way of safely estimate their load on the systems, which makes a priori estimates of parameters such as failure rates and WCET impossible. The only way to approach these kinds of systems is with measurements. Examples of architectures that make such estimates tough are interrupts, cache, and bus arbitration. This fail-aware approach has as a goal to solve non-maskable performance failures. The fail-awareness is defined as a best-effort approach to construct fail-safe distributed hard real-time applications for *partitionable* systems. A partitionable system is a system where sets of processes can split into disjoint subsets due to some failure or error.

The assumptions for this fail-aware approach are that the processes have their own hardware clocks which are linear in real-time, and independent of the loads on them. By using these clocks, the system can have other means of synchronization other than acknowledgment from each other and we can move away from a "completely synchronized system" into a "timed asynchronous system". The first of these two models are a more ideal model where real-time delays of processes and messages are all known a priori, and upper bounds for failures per time unit are known. The other has no given upper bound for failures per time unit and is a more realistic approach when used in actual systems and makes use of the hardware clock for communication and actions.

Fail-awareness is to require that all failures that occur are maskable as long as the number of failures per time unit is below some set threshold. When the number of failures rises above this threshold, the failures must be handled by some specific exception semantic. The important part here is that a fail-aware server needs an indicator to be able to tell when this threshold is broken and must be able to signal this to the client. The switch from a normal synchronized specification to a fail-aware specification is to have such an exception indicator.

When a partitionable system creates a new partition, fail-aware services should be provided to handle this partition as close to an ideal synchronized partition as possible. The services are a set of hierarchical calls that creates a safe way to processes in such a partition to communicate with each other. This new handled partition is called a logical partition and provides atomic broadcasting by means of message forwarding and removing unnecessary connections. [20]

### 2.2.2 Available fail-safe systems

A fail-safe system is often described in the way of redundancy, like in the article [18]. Here a duplex system is used with two fail-safe units within. An example of an area of use is a railway system where the briefest interruption could be fatal, and therefore the safety is key. But this example like many others needs to have some sort of support for a fault-tolerant computing system as this is desirable to satisfy dependability requirements, availability expansion, and good maintainability. The goal of the article was to increase availability without degrading the safety of a system. The technique is that the system works with two units that can both provide full system functionality by switching to the other when a failure is detected while the one that fails will go into a fail-safe state.

### 2.2.3 Fail-safe motion planning of autonomous vehicles

The fail-safe approach has of course many areas of use, where an important one is within motion planning for autonomous vehicles. In [32], one fail-safe approach for vehicles meant for personnel transport is explained, where factors like comfortable transport and safety are important. This motion planner is based on planning the path based on both natural trajectories from roads and a vehicle in front. Since the transport need to both be safe and comfortable, all possible trajectories are considered, and there is a goal to choose the ideal one for these conditions. The motion planning is first done by planning against a leading vehicle and choosing an ideal path based on their estimated trajectory, then an emergency action, a fail-safe, is planned to bring the vehicle to a standstill. At the moment the vehicle cannot find what it considers a feasible path, the emergency maneuver is activated as the fail-safe option. At all times, the vehicle must have a fail-safe state as an option.

## 2.3 Dynamic deployment of real-time tasks

The general hard real-time theory is often considered both rigid and strict with unchanged theory from the early 60s. With the development of modern computing systems more dynamic solutions have become available and give options for new types of solutions. **Dynamic deployment** of real-time tasks is the concept of being able to solve and distribute real-time properties during the run-time of a program[4]. With dynamic deployment, there might be other factors that play a role when deciding a scheduling strategy than what one is used to with classical hard real-time theory. This section will show research done in fields where dynamicity is implemented as a part of the strategy for scheduling a real-time system.

### 2.3.1 Deadline miss ratio constraints

An extensively used method for soft real-time scheduling is by using a deadline miss ratio. The method involves allowing deadline misses up to a ratio, and if this constraint is breached the system would go to failure. Statistics is used with best effort-strategies to keep the deadline miss ratio to a minimum and below the threshold, like in the article [33]. Here the execution time of a task is approximated with a distribution function and then checked for constraints to deadline miss ratios. The thresholds for deadline misses can be fixed to fit a systems requirement or to the criticality of a single task (a deadline miss threshold of 0 would equal a hard real-time task). The approach makes use of optimization techniques for execution times of tasks, as a fixed approach often is too naive for reality.

The goal is to optimize a cost function which is based on mapping tasks to a designated processor element and assigning a priority to it. By heuristic methods for mapping and assigning priority, feasible scheduling under deadline miss ratio constraints and priority is given with analyzing of the cost function. The environment is a multiprocessor, which

[4]This definition makes this a gathering term for dynamic decision making during run-time but can be inconsistent with other definitions of the same term. In fact, the definition of dynamic deployment varies a lot with the context it is used in and no true general meaning can be found.

makes the algebra and system description very complex. A lot of statistical mathematics is also necessary to provide the correct proofs for this approach.

### 2.3.2 Power constraint systems

An interesting set of dynamic real-time scheduling is power constraint systems. The goal in these systems are not to ensure better-performing systems with regards to time efficiency, but rather to minimize the power consumption of the system. This is very relevant for low-power embedded systems where real-time constraints are crucial. The problem here is that the power usage is linearly dependent on the time and quadratically dependent of voltage, which eventually leads to that reducing the speed of execution of a task will cause them to consume less energy, but take longer time to complete [35]. The article [29] describes an approach for a hard real-time system, which wants to keep the power consumption to a minimum by interchanging between two modes of execution: high voltage, **HV** and low voltage, **LV**.

First, an off-line analyzis is done where every task, their WCET, and their deadlines are assumed known, and a feasible schedule is made for when they are to be executed. The tasks get their HV and LV executions set, and the most power-efficient schedule is set up. The off-line phase is done by simulating the task set before execution, and during execution, another analysis is done to further cut down on power consumption. The online phase is based on the opportunity for that a task finishes before its WCET, and compares the unfinished work previously found in the offline phase to the actual unfinished work online. When the processors take up another task, the expected unfinished work is compared to the actual unfinished work, and the scheduling will use LV modes of tasks until the actual unfinished work equals, or is bigger than the expected unfinished work. With this logic, the scheduler will run as much as possible with the low power version of a task and only use HV when found necessary.

By using such an approach, one can already assume that there exists some way to schedule the task set, i.e. the CPU is already known to be powerful enough for the task it will be given. The question is rather to find a more power-efficient way to schedule the system.

### 2.3.3 Dynamic scheduling

One way of approaching dynamic real-time is shown in [28]. This article describes an approach where during run-time, the system will change between two different scheduling strategies. This is in a soft real-time system, and the criteria for choosing a strategy depends on deadline misses and load of the system. During underloaded conditions the strategy used is the dynamic earliest deadline first, **EDF**, strategy because this is both optimal and efficient. EDF suffers however of exponentially decreasing performance when a system gets overloaded. Therefore, a strategy that is shown to provide good results for overloaded systems is changed to, the ant colony organization, **ACO**, algorithm.

The criteria for changing the active algorithm depend on deadline misses. The system will initially start with EDF, and use this algorithm until a single deadline miss occurs. Then the ACO algorithm will be used until the overload of the system can be considered dealt with. The article suggests that this could be after 10 deadlines are met in a row, but it

is reasonable that the number of deadlines can be any context relevant number of deadline hits.

### 2.3.4 A Technique for Adaptive Scheduling of Soft Real-Time Tasks

Many types of real-time systems require management of overload situations. These systems are in the category of soft real-time systems as deadline misses can occur, but they still need to be handled with care and consideration to ensure the correct functionality of the system. The article [5] describes a way to use adaptive scheduling for such overload management.

The described method is based on the possibility for soft real-time tasks to choose, rather than to be assigned, a period within some closed interval. Shown in equation 2.1 for a soft real-time task i, where its corresponding period T.

$$T_{i,min} \leq T_i \leq T_{i,max} \tag{2.1a}$$

$$T_{i,min} \neq T_{i,max} \tag{2.1b}$$

The adaptive scheduling methods in the article are based on choosing rates for tasks within its interval and using static rate monotonic, **RM**, scheduling based on the utilization of the final system. The utilization ($U(x)$) has as a condition to be below the available processor utilization for soft real-time tasks in an overload condition ($\Delta_{ov}$), as described in equation 2.2.

$$U(T_s) \leq \Delta_{ov} \tag{2.2}$$

Here $T_s$ is the subset of periods for soft real-time tasks, this approach can allow tasks to be hard real-time tasks which periods are not changeable and set on beforehand.

In the approach, the problem is formulated as a linear programming, **LP**, problem where the variable is the period of a task ($x_i$), and with the possibility to assign weights to these tasks ($v_i$) (see figure 2.3). The straight use of an LP approach is however possibly very computational heavy, especially if required to do on-line, and the solution space can turn out really big. Instead of direct LP approach, several alternatives ways to obtain adaptive scheduling of these soft real-time tasks can be used.

$$max \sum_{j=1}^{s} v_j x_j \tag{2.3}$$

The first approach is by using **Greedy Algorithm** approach. This is to divide all the tasks into two subsets, one with their maximum period chosen and one with their minimum period. The idea is to check total utilization by these chosen subsets and check for when 2.2 no longer holds. In the beginning, all tasks are chosen with their max period, then by iterating one by one, the minimum period is chosen for a task, and at the moment the 2.2 no longer holds, the subsets are chosen.

More general **Saturation Approaches** uses more delicate ways of deciding tasks to be set to their maximum period. Here other factors such as relaxing and scaling are used.

With **Value-Based Heuristics** the task order is looked into in order to use the greedy approach with more efficiency. Also, by changing pre-set priorities the saturation approaches can be improved.

The tasks can be assigned with nominal, which can lead the system into an overloaded state. By using these nominal rates of the tasks, new rates can be chosen based on the minimum distance which still gives feasible scheduling. The problem will then go from an LP problem and turn into a quadratic programming, **QP**, problem. Other QP approaches are also discussed such as choosing new rates to turn the tasks as close as possible into harmonic functions. These QP approaches are NP-hard.

Chapter 3

# Theory

The main purpose of this chapter is to provide background information relevant to important themes discussed in this thesis. Sections 3.1 and 3.3 will try to contextualize the subjects for discrete computing and real-time programming, while more detailed description will be given in section 3.2. The content is a mixture of basic theory within the fields of linear system theory, control theory, real-time programming, and signal processing.

## 3.1 Control theory

The subject of control theory revolves around getting some process[1] to follow a specified behavior in a safe and desired way. The processes are usually affected by unpredictability and dynamic behavior such that some sort of control is needed to ensure correct behavior. The entity to be controlled is called a **system**, and the entity that controls the system is called a **controller**. The system will interact with its **environment** which is the near physical world that the controlled system operates in.

With dynamic control comes the need for swift and accurate calculations at the correct time. This makes the use of real-time programming very relevant within control systems with many natural connections. Without diving too deeply into the basics of control theory, this section will look into some basic properties of analysis of a control system while addressing them from a real-time point of view[2].

### 3.1.1 Types of control systems

Although one can argue that almost no two control systems are the same, one can roughly gather control systems in three different classes:

1. Monitoring systems

[1] The definition of "process" is vague here and could mean any set of actions that requires control.

[2] Lines drawn between control system and real-time programming are heavily inspired by the content of [12]

2. Open-loop control systems
3. Feedback control systems

A **monitoring system** does not interact with the environment it is operating in, but rather gather and saves information about it. Possible applications for monitoring systems are air traffic control, radar tracking, and alarm systems. This information is retrieved with different kinds of measurement tools and sensors. A lot of sensor data is acquired in periodic time intervals which differs from one sensor to another, this could be carried out as a set of concurrent periodic tasks in a real-time kernel. In the case of sensors that acquire critical information, a hard real-time kernel must be used to ensure the correct behavior of the system.

**Open-loop control systems** interacts with its environment without being dependent on its state. This means that actuators of the system is not strictly interactive with feedback from sensors and can perform actions when any plan is ready. Open-loop systems are typically making a plan based on its information and execute it without any online interference and therefore does not need real-time computing.

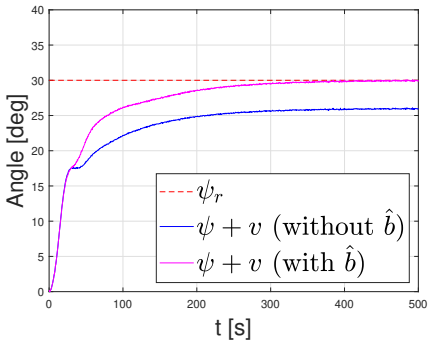
**Feedback control systems** are mixtures of the two previously mentioned classes. They make use of sensors to acquire the state of the environment, compares it to some desired reference functionality, and makes interactions to influence the environment in the right direction. In other words, there is a feedback loop from the environment, through the controller and back into the environment again. Now there is not just the sensor acquisition that needs to be carried out with real-time computation, but also the interaction with the environment. Real-time computing is essential to ensure that sensor data is carried out at the correct time for the controller to output the correct action. A good example of how critical it could be that these timing constraints are met are with flight control using an autopilot system. A late reading of the height the plane is at could cause the airplane to stall in a critical situation which may lead to a very dangerous situation for all passengers.

### 3.1.2 Stability and response

There are many analytical methods to judge the behavior of a control system, but the most important of all is **stability**. An unstable system will give an ever-growing output and most definitely make sure the system goes to failure, so to check for stability of a system is always a necessity for any control system. Control theory offers numerous ways to check if a system is stable by analysis of transfer functions, poles, properties in a bode diagram etc. To ensure a stable system is a necessary condition for the functionality of a control system, but it does not necessarily tell if the behavior can be considered good or not.

After the stability is ensured, one way to determine the quality of the behavior is to analyze the **response** of the system. The response in this case is meant by any signal in the system that is influenced by the controller and is relevant to the control system. A step-response is to apply an input to a controller that moves from 0 to a given step immediately after time 0. Often is the step-response a good indication of the behavior of a system as it shows the damping of the system (oscillation about the reference value) and how fast the reference is reached. A growing output value and big oscillations indicates a system that is near-unstable or unstable. A step-response is normally considered good when the reference is reached quickly without too much fluctuation from the reference value.

**Figure 3.1** Example of the step-response of a compass course for an automated ship (see chapter 5).




Responses with great oscillations can have critical consequences in a physical system where fast changes will normally be harder to perform than in simulated graphs. It can also be very challenging for the wear and tear of mechanical actuators like motors and breaks.

## 3.2 Discretization of continuous functions

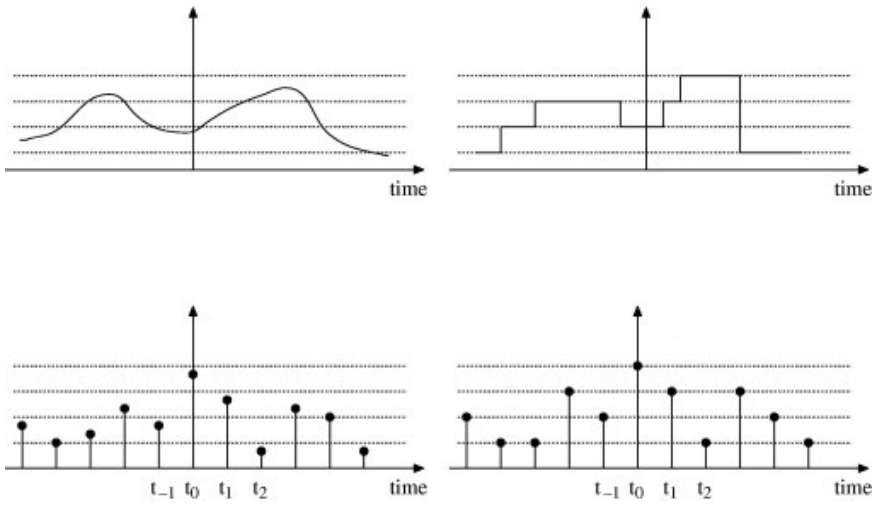
When working with theoretical control systems it is common to assume continuous signals into a continuous controller, and while this assumption may be sufficient for many real-life systems, the reality is that modern computers are based on digital design. A digital design requires discrete data, which again means that it can only work with a given number at a given time. This means that to be able to work with continuous measures like wind and waves, the signal must be **discretized** before a digital computer can comprehend its values. After a computer has comprehended the discrete data, the signal may be made continuous again if the receiver of the output requires it. In this section, the focus will be on basic concepts for discretization as well as common methods for how discretization is performed[3].

### 3.2.1 Discrete vs. continuous signal

Discrete and continuous signals are differentiated with **discrete-time** versus **continuous-time** and **discrete-valued** versus **continuous-valued** [36]. Discrete-time signals only assign a value to distinct points in time and have no information apart from these points in time. A continuous-time signal has a value assigned to every point in time and could make a continuous graph. Both discrete-time and continuous-time signal can be **discrete-valued**, the signal only takes values from a finite set of possible values, and **continuous-valued** where the signal can take all possbile values in a finite or infinite set.

[3]This section is inspired by and builds on the content in [4].

**Figure 3.2** The four types of signals represented by a sine curve: upper-left - continuous-time, continuous-valued, upper-right - continuous-time, discrete-value, down-left - discrete-time, continuous-valued, down-right - discrete-time, discrete-valued [46]




### 3.2.2 Properties and rules

#### Sampling and sampling frequency

To convert the continuous signal into a discrete signal it is important to convert from continuous-time data to discrete-time data, this is done by the concept of **sampling**. Sampling is the process of reading the value of a continuous signal at a certain point in time, making it into single-point value with a coherent time-step. How often these readings are done is called the **sampling frequency** $f_s$ and the interval between two readings is called the **sample time-step** $T_s$. The sample time-step can be constant, called a **fixed-step** method, or it could vary due to some property, called a **variable-step** method. The sampling frequency is inversely proportional to the sample time-step:

$$f_s = \frac{1}{T_s} \tag{3.1}$$

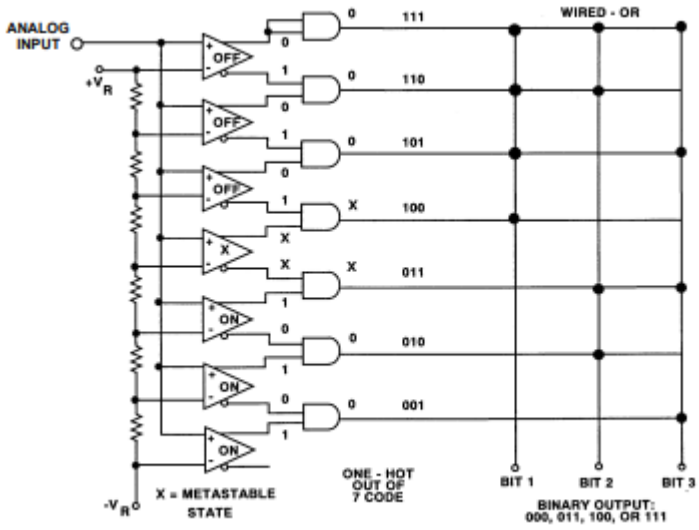
and will also be constant when using a fixed-step method.

A well-established method for creating analog signals from measurements is to create a voltage signal which corresponds to the value of the measurement [22]. This method is being used for many types of measurements such as temperature, weight, and the fluid level of a tank. Let us take the electrical temperature sensor Thermistor, in this sensor, there is a constant current flow and a resistor which will vary with the temperature. In a PTC Thermistor the resistance increases as the temperature rises, and according to Ohm's law leads to a higher voltage signal [44].

### ADC

To create a digital signal from these voltage based sensors a digital-to-analog converter, **ADC**, is used. An ADC samples the voltage signal and appoints it a digital value at each sample. There are multiple design options for these converters where many of them are based on comparing the input voltage from the sensors to a reference voltage in the ADC. The **resolution** of the ADC is the number of digital levels possible to dedicate the analog voltage signal. Essentially, this is decided by the number of comparators in the ADC, and each comparator represents one bit of a binary word. So if an ADC were to have N comparators, the available levels, and therefore the resolution of the ADC, would be $2^N$. A higher resolution will give a better representation of the analog signal because the voltage gap between one level represents the margin of error and would be smaller when increasing the number of steps. [25]

**Figure 3.3** Standard 8-3 ADC comparator. The input voltage is compared to a reference voltage which declines with each comparator. The 8 comparators will make a digital 3-bit word.



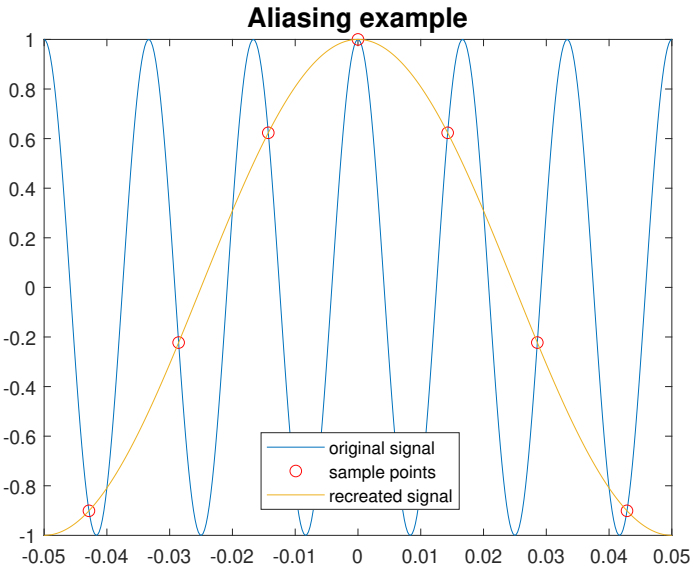

### Shannon-Nyquist theorem

When an analog signal is sampled it must be possible to correctly recreate it based on the samples. Because of this, the sampling frequency must be high enough to ensure that exactly the correct signal is picked up and is indistinguishable from others. The Shannon-Nyquist theorem, also known simply as the **sampling theorem**, states that the sampling frequency must be at least twice the highest frequency of the analog signal it is sampling, $f_{sys}$.

$$f_S \geq 2f_{sys} \tag{3.2}$$

Failing to do so may cause the construction of a signal different than originally sampled, this phenomenon is called **aliasing** and is presented in figure 3.4. Essentially would a higher frequency make for a more correct representation of the analog signal.

**Figure 3.4** Example of aliasing as a result of too low sampling frequency.



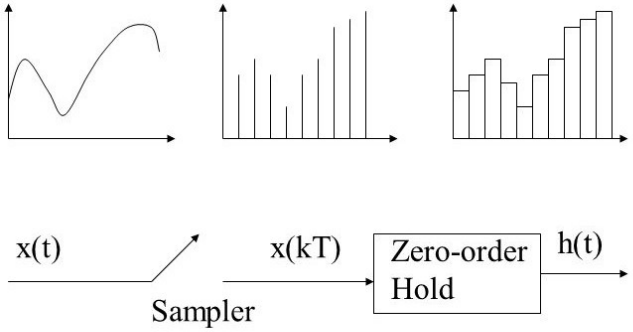

### 3.2.3 Methods of discretizaton

While sampling and ADC are physical entities for the discretization of continuous signals, other methods are required to analyze the discrete signal in the form of an algebraic function. Such methods are very useful for simulation of systems, and programming languages such as Matlab can give a wide selection of different approaches. Depending on the method of how the discrete signal is treated, the algebraic function will change. Two approaches for dealing with discrete signals will be explained here.

#### Zero-Order-Hold (ZoH)

The zero-order-hold method is the simplest and most intuitive method for dealing with discrete signals. It gets the discrete value from the discrete system and holds that value until it gets updated with a new value. The ZoH method can work as a direct discrete-to-analog converter (DAC) or together with a sampler as a holding element to create a continuous signal in discrete time (see figure 3.5). The resulting output is a stair-like continuous signal which can be used by continuous functions and methods.

**Figure 3.5** Use of ZoH in a system. A continuous signal gets sampled, and the zoh makes a continuous signal in discrete time. [8]




### Z-transform

Using ZoH is essentially to treat a way to treat a discrete system as it was continuous, but this approach is not always sufficient. The z-transform is a method that performs an exact mathematical analysis of discretized continuous systems. When a continuous function, $f(s)$, and the sample time $T$ is known, one can use the z-transform to find an exact discretized equivalent which again can be analyzed. The transformation is very reminiscent of the Laplace transform. It is used on systems written in the frequency domain, where the frequency variable, $s$, is changed out with the discrete variable $z$.

$$s = \frac{2}{T}\frac{z-1}{z+1} \tag{3.3}$$

Then the z-transform is used to get the function from the discrete frequency domain to the discrete time domain.

$$f(z) = \sum_{k=0}^{\infty} f[k]z^{-k} \tag{3.4}$$

With this transformation it is possible to check for desirable properties like poles and stability.

### Other methods

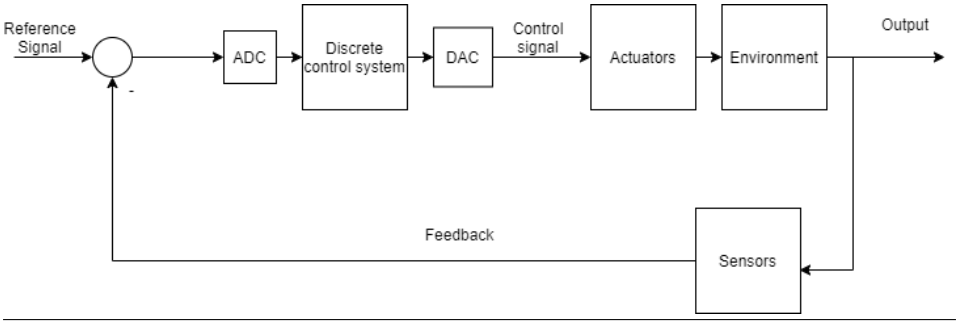
There are many other ways to discretize a continuous signal than ZoH and z-transform. The **w-transform** is a further development of the z-transform and can offer a closer resemblance to the continuous frequency plane, making analysis of stability and poles easier as the approach would be very much the same. Matlab have a discretization function called *c2d()* which discretizes a continuous function. The standard method is the ZoH, but it also offers the use of:

- FoH, triangle approximation
- Impulse invariant discretization is the same as the z-transform, only that Matlab scales it by T.
- Bilinear Tustin method
- zero-pole matching method, not much used and not that good. It has problems preserving information.
- least-squares method

### 3.2.4 Discrete control systems

Modern control systems are realized with time-discrete computers as it is far superior to any analog computer in regards to computing power and efficiency. This is an important consideration to make when designing a control system as the implementation may differ from the theoretical system. A discrete control system will still need feedback from analog sensor signals and give output to analog actuators such as motors and steering devices and makes use of DAC and ADC.

**Figure 3.6** Block diagram of a general discrete control system using ADC and DAC.




A discrete control system can be achieved by approaching the system as it was continuous. This can be done if the sampling time $T_s$ is smaller, or at least in the same size order, as the fastest dynamic of the physical process. The discrete controller turns the continuous formulas into recursive algorithms as this is a natural way for a digital computer to work. This can be done by interchange the time/frequency variable with the **time delay operator** (see figure 3.3) and transform the formulas with z-transform. A discretization can also be approached using a simple **holding element** to get a signal from continuous-time into discrete, which can be implemented into a continuous function by adding a simple time delay equal to the sampling time of the holding element.

## 3.3 Discrete Kalman Filter

The **discrete Kalman Filter** is a well-established tool within control theory for closed-loop control systems where measured signals contain disturbances and noise. It was introduced

already in the early ’60s when the development of digital computers had reached a state where it could be implemented into real-time application [10].

### 3.3.1 Filter end estimator

The Kalman filter is also called for linear quadratic estimation as it uses linear algebra to combine the measured environment-state and state-estimation to create the output state. This means that the output state from a Kalman filter will not simply be a filter for the input state as the algorithmic execution will attempt to create a *better* signal than that. When a Kalman filter is working as intended it will correctly cancel out disturbances which would mislead a controller to compensate for something it should not. For this to be possible, the Kalman filter assumes that disturbances are on a zero-mean Gaussian form to get the dynamic system on a linear form.

### 3.3.2 The Kalman filter cycle

Discrete Kalman Filters is based on a recursive algorithm that executes at every time step with a predetermined frequency. The steps of the Kalman filter at each of these time steps are:

1. Predict current state
2. Compute Kalman matrices
3. Update output
4. Project ahead for next computation

[4] The first thing done in the Kalman filter algorithm is to predict the current state. This is done by fetching estimations made at previous computation and inserting the current time step. This estimation will be used further in this time step.

The matrices in the Kalman filter algorithm is computed by making use of both measured state and estimated state. "Matrices" here means that an estimated covariance matrix is eventually found by multiple steps of calculation using the previous step estimated covariance matrix and covariance matrices for the process and observation noise.

The computed inner matrices of the Kalman filter is used to update the output estimate. Now, this can go directly as the feedback for a controller which can make sure the reference state is followed.

The final step of one computational cycle of the Kalman filter is to estimate the state and the covariance matrix which will be used again in the next computation.

[4] The Kalman filter is generally recognized to have only two steps: predict and update. The extra steps added here is a further decomposition of the two mentioned steps.

### 3.3.3 Update frequency

Kalman filter was introduced around the same time as digital computers, and its discrete nature fits perfectly for its recursive algorithm. A continuous signal, as most signals in a realistic control system are, would need to be discretized before the Kalman filter can use its information. The sampling of the continuous signal does need to follow the principles of the sampling theorem (see section 3.2), but the frequency of the Kalman filter algorithm technically doesn't. However, more frequent cycle execution of the Kalman filter will cause smoother output signals with better filtering of unwanted frequent disturbances. So, although not necessary, it is common to choose the same frequency for the discretization of the input signal as the update frequency of the Kalman filter. A higher frequent update of the filter would not make sense as the input state would stay the same until a new sample was made, a lower frequent update could possibly lead to some weird aliasing-like behavior.

Chapter 4

# Categorizing of domain knowledge in real-time systems

Real-time systems separate from normal computing systems by containing stricter constraints, and generality is therefore more comprehensive to achieve. A real-time system does also often applies to a lot of more specific designs than normal computing systems and could benefit a lot by knowing what its surroundings are. The concept of making use of information of the environment around a specific system sounds reasonable in itself, but the questions one has to ask is; for what can this information be used, and in what realistic cases can this information prove itself useful? In this chapter, the term **domain knowledge** will be defined and categorized into different classes within real-time theory. Within these classes, there will be suggestions on how to utilize domain knowledge for mitigating the current drawbacks of real-time system.

## 4.1 Defining domain knowledge

When talking about domain knowledge, it is important to define exactly what is meant by a "domain" in this thesis. The terminology determined in from 3.1 will be used, and the definition of a **system** is extended to any physical device that has to execute in some desired behavior. For a computer program, this desired behavior could be to provide the correct computations. A **real-time system**, however, does not only need to provide correct computations but also has constraints to at which times these computations are provided. A real-time system could be a ship following a path, a robot following orders, or a power plant that must be kept stable. The **domain** of a system is the external environment it is operating in and is a collective term for every external entity which can influence the behavior of the system. This means that the domain of a system is broader than just pure physical entities and includes conceptual and architectural influence of the system. For example, the domain for a microcomputer inside a car would consist of physical entities such as wheel rotation counter, but also its programming language and its CPU architecture. For a control system,

like a monitoring system for air traffic control, the domain would contain its environment.

The domain is important for any system as it needs to be considered by any system design, whether it is built by generalized solutions or not. On the other hand, the domain of a system could also help out the system design as it may contain information that could be beneficial when designing the system. Any kind of information a domain can offer to its system will be defined as the **domain knowledge**. This also includes how a system is able to handle things like deadline misses, overload, and different scheduling strategies. In control systems, for example, the state of the environment provides domain knowledge in itself. In other words, the domain knowledge can be obtained by both utilizing already existing information and to extract new. A **condition** for a system is a situation that based on the domain knowledge can be determined somewhere between easy and challenging. An easy condition requires less strict timing constraints for ensuring correct behavior while a challenging condition might require smaller periods and tighter deadlines. To analyze the current condition of a system, based on its domain knowledge, might be a key to move away from the pessimistic hard real-time approach towards more efficient solutions.

## 4.2 Control systems

In control systems, there is a huge potential to utilize domain knowledge for alternative ways of scheduling a real-time system as there is a lot of information to retrieve by default. Whether it is a closed-loop, open-loop or a monitoring control system, the state of the environment is essential for basic functionality and can be considered as the domain knowledge. The state of the environment gives inputs to the control system so that it can react to it and apply the correct modifications to the environment. The real-time system makes sure that the periodic update of the state variable is sufficiently often enough to correctly reconstruct the state to the control system, and that the output of the control system is correctly computed and provided at the right time to the environment. Sensor measurements from the environment and interaction from the actuators on the environment are all treated as concurrent periodic real-time tasks, and their update acquisition and update frequencies define their deadlines and periods. If it is crucial to guarantee the correct behavior of the system a hard real-time kernel is used.

The usual approach for these kinds of systems is to analyze the task set off-line and guarantee a feasible schedule with a given scheduling strategy using the imposed timing constraints. These timing constraints must often be based on WCET to guarantee a feasible schedule and correct behavior of the real-time system. However, as the environment for many control systems are very dynamic and the needed update frequency of the system varies a lot dependent on the situation, using the WCET can be a very inefficient approach. For example, an autonomous ship which job is to carry equipment across a short distance might be working in normal weather conditions for most of the time but must be prepared for tough weather conditions at any time. This means that time constraints for a real-time system must be defined for the most challenging situations, which might just happen a small proportion in its working time. By analyzing the state of the environment, it is possible to conclude how challenging the conditions are, and new options for real-time scheduling opens up.

### 4.2.1 condition-based behavior

When the controller system meets a challenging condition, some predetermined action can be activated to make the controlling of the system easier. For example, when an autonomous drone meets a lot of wind it will be beneficial for the reference speed to go down. When the reference speed goes down it will become easier for the controller system to meet this reference, and more CPU can be dedicated to meet the wanted reference course. On the other hand, when the conditions are fairly easy, for example, no wind influence, we can allow the drone to increase its reference speed as it will become easier to keep the wanted course. This is applicable to any autonomous vehicle where the weather influence is highly variable. Then, by making decisions that make the controlling of the system easier based on condition, the time constraints do not need to be as strict as when expecting the same behavior independent of the condition. The CPU computation power can be scaled down accordingly. This does demand that the system is allowed to change behavior, and maybe lowers the QoS, but for many systems this is reasonable.

### 4.2.2 Variable frequency update of control systems

When the controller system cannot allow such changes in behavior, an alternative way of lowering the use of CPU is by a variable frequency update. The thought behind this approach is to still check the condition of the environment and lower the CPU usage when possible while still keeping the correct behavior of the system. As mentioned earlier, the update of a controller is treated as a real-time task where the update frequency matches the period of a periodic task. By analyzing the state of the environment, a condition can be found, and based on this condition a frequency for updating the controller can be decided. Variable frequency update of controllers in itself is not a new topic, and have been a researched field for a long time [17] [39].

There are multiple ways to implement a controllable frequency update of a control system. As the relevant domain knowledge here is the state of the environment, the update frequency of the sensors that measure these values should be held constant, and rather the actual update of the controller can be dealt with in a variable way. This can now be treated as a sample-data control system, and the theory for adaptive sampling frequency can be applied [17]. As long as the correct constraints are applied to the variable frequency, the stability of the system can be guaranteed [39]. By applying domain knowledge as an indicator of how often the controller needs to be updated to execute correct behavior for the system, this has become a very viable option.

### 4.2.3 mode-based control systems

Although the mechanics of a variable update frequency of controllers are very appealing as the continuous state of a controller can be preserved, this is a very non-linear and complex approach. To check and guarantee the stability of such systems is not an easy approach and requires a deep understanding of signal manipulation and advanced control theory. This approach could, however, be simplified if instead of continuously deciding what the update frequency the controller should have, the controller simply changes between a given set of frequencies. This approach will be called a **mode-based control system**, and each pre-set

frequency will be considered a different mode. Modes in controllers is an exciting area of research and is represented in topics such as *sliding mode controllers* [14] [47] and *switch control systems* [37].

When the frequencies are decided in advance, an offline analysis can be done for each mode, and both stability of the control system and a feasible schedule for a real-time system can be guaranteed offline. The idea is that much like in the variable frequency update, the domain knowledge of a system can be used to find what kind of condition it is in. In easier conditions for a control system, the controller can choose a mode with lower frequent updates and free up CPU time, either be dedicated elsewhere or simply to save power consumption. The domain knowledge will be used as a decision-maker for when the controller can/should change between the pre-set modes. This can be a question of when it is acceptable with less reactive controlling or when it is crucial that another user of the same CPU needs more power. An in-depth example of how domain knowledge is utilized with this approach can be seen in chapter 5, where an autonomous ship analysis the weather conditions to decide how reactive its controller needs to be.

The **transient periods** between a change from one mode to another is the most challenging situation for mode-based controller systems. If the differences in frequencies for different modes are big, an immediate change might bring an unexpected response or a reaction that a physical system might not be able to follow. This method is a suggestion by this thesis and the actual problem with transient periods not much researched or tested. But as each mode defined for these systems must give a stable response, it is reasonable to believe that every frequency in the interval between two stable frequencies will also give a stable system. This does assume that no frequencies out of this interval is touched, which could be guaranteed with a strictly increasing or decreasing change. A sampling frequency can only turn a control system unstable if the continuous system itself is unstable or if the sampling frequency is lower than the sampling theorem allows (3.2.2).

### 4.2.4 Autonomous vehicle control

When it comes to feedback control systems, there is perhaps no better example of its area of use than with autonomous vehicles. These vehicles are becoming more and more popular and are by many considered the future of transportation. Whether this is autonomous cargo ships, flying drones or self-driving cars, there is no doubt that a lot of big industries show interest in this field. There is a lot of dynamic variables when it comes to autonomously maneuver a vehicle, and both quick and correct computations are in order to ensure correct behavior. This makes the use of real-time programming a good solution for the control system. These vehicles are often equipped with additional devices such as cameras and radars which have their own purposes. These devices may have a purpose towards the maneuvering of the vehicle, such as collision avoidance of other vehicles, or something unrelated such as seafloor monitoring.

Within the theoretical design of autonomous vehicles, it is normally assumed continuous controlling and there is very little talk of physical limitations such as CPU and power consumption. Therefore, to ensure as correct behavior as possible, these vehicles get equipped with multiple CPU cores with more than enough power. Although unclear of this has been a problem or not it is reasonable to believe that as this technology grows more complex, there will be a problem meeting CPU demands in the future.

Video cameras onboard autonomous vehicles will often make use of image processing techniques to execute their purpose. When this is crucial for the correct behavior of the vehicle, for example, to avoid collisions with other vehicles in the course, the timing of the results is crucial and it makes sense to use real-time programming for the image processing system. Since it can turn fateful to not get the results on time, a hard real-time programming approach is in place. With hard real-time tasks, there is a need for good WCET estimates. The computation time for image processing is highly dependent on the resource it is analyzing and can vary a lot. When WCET analysis is done statically offline, which is the norm, tasks like image processing can suffer from very pessimistic WCET estimates for rare conditional computations. By applying domain knowledge, the system can now decide when the rare conditions are likely, and setup the hard real-time scheduled system only when necessary. This frees up a lot of CPU in normal conditions, which can be dedicated to other tasks. There is now at least a bigger chance of collecting multiple systems to a single CPU and reduce the computational need for the vehicle.

### 4.2.5 Examples of autonomous vehicles where domain knowledge is applicable

#### Surveillance drones

Flying quadcopter drones have been commercialized in recent years and have revolutionized the film industry with its simplicity. One example of this is an autonomous drone that uses a built-in camera for surveillance of power lines [30]. The pictures from the camera use filters to remove measurement and environmental noise by using characteristics of power lines in pictures in its advantage, and then a detection algorithm is used to find and analyze the state of the power line. In [30] the filter is based on a pulse coupled neural network (PCNN) and detection done with a Hough transform.

For this particular control system, domain knowledge can be acquired by measures of wind and pattern from the camera. By using this information both condition-based behavior or a form of variable frequency control can be applied. With condition-based behavior the drone could slow down when the power line is hard to detect, giving the image processing naturally better time to process its pictures. Likewise, when it is easy to distinguish the power line from the background, like in fields or forests, the speed can be pushed up for more efficient work. The same argument can be done for a constant pace but varying how much CPU dedicated to the image processing by dedicating more when the power line is hard to distinguish and less when it is easy.

If these systems are using the same CPU, this will also influence the control of the drone and its disturbances will also have to be taken into consideration. This means that with a variable frequency approach the CPU has to contain enough computing power for a worst-case scenario when the condition is tough for both the image processing and the control system. But when it is not a worst-case scenario, the system would be able to save power consumption by dedicating less CPU when the conditions allow it. This approach is also applicable if separate CPUs are used for the two systems, and the implementation might be less complex as there are fewer variables to analyze when deducting a condition from the domain knowledge.

**Autonomous ships with collision avoidance systems**

**Autosea** is an NTNU student project whose goal is to implement collision avoidance onboard an autonomous boat using two cameras [9]. The cameras provide pictures to a computer vision process which goal is to check for obstacles in its course with a detection algorithm. When an obstacle is detected a collision-avoidance plan is made and given to the vehicle control system which will maneuver around it.

A collision avoidance system using lidar and radar onboard an autonomous ship with underactuated dynamics is suggested in [43]. The ship is limited to "unicycle" dynamics which limits the independent control of the degrees of freedom. Three different systems are implemented for correct control of this ship:

- Guidance control system - have as a job to keep track of the position of the vehicle and to create references for the vehicle to keep the correct course.
- Collision avoidance control system - this system uses radar or a lidar as sensors to discover hindering objects in the path of the vehicle. If such an object is discovered, its position (and possibly its velocity) will be read and the system checks if it can lead to a collision. If it does, a safe path away from the object is created for the vehicle to follow.
- vehicle dynamic control system - uses references given by the guidance control system to move the vehicle on the right course. Control engineering techniques are used together with the system dynamics in the form of differential equations.

Although these ships are using different technologies for the detection of obstacles, they are conceptually similar with the same goal. Both divides its functionality into different systems which will need to communicate with each other. In terms of real-time programming, this is an interesting case as to whether or not these systems should be implemented in the same scheduler or considered independent of each other. Let us say these systems are implemented on the same CPU core, this could lead to less complex communication architecture than with separate due to memory management and shared buses. Fewer CPU cores might also be both cheaper and less power consuming. Domain knowledge could be applied for condition-based behavior and further CPU usage could be shaved down. With a variable frequency update control method, there is an opportunity for delegating more CPU if a system has tougher conditions than the other.

## 4.3 WCET analysis with online measurements

### 4.3.1 Defining a general framework

As described in section 2.1, alternative ways of analyzing WCET for real-time tasks have already been widely researched, and many good approaches already suggested. With reference to some of this research, some general categories within WCET estimation will be suggested based on the approach used for analyzing. Hopefully, this will show that the term *system-specific solution* is not as specific as first assumed and that there is some generality to find for future systems. To utilize online measurements for a WCET analysis

is a popular approach ([3] [23] [26] [40] [42] [41]) and although different authors have their say in the matter, the framework for these approaches are more or less the same.

When a general framework can be established for how to handle WCET analysis, the domain knowledge for a system can be used to fill in how to implement this framework for a specific system. With WCET, the relevant domain knowledge will be the hardware that runs the system to establish what tools of measurement are, and what kind of programming language the task is running on. In other words, there is no actual need to dynamically collect domain knowledge for this type of approach. The framework is described in many different ways but could be summarized in three steps: partitioning the task, generate test data and measure the partitions, calculate the final WCET. This does however assume that a partitioning algorithm is available for the given system, which in many cases is a bold statement. There are well-established methods for this kind of partitioning such as evolutionary algorithms [40], CFG partitioning [42] various model checking approaches [24] [45]. However, dynamic measurement methods such as these are fairly complex and not easily implemented compared to more traditional static methods.

### 4.3.2 Safety of a WCET analysis

A big advantage of using offline analyzing methods is due to the safety of their results. Because of the pessimistic nature of these estimates, they will never be shorter than the actual WCET, which is one of the reasons why this approach is so widely used. However, the static analysis gets increasingly harder to implement with the complexity of the program it is analyzing [2]. This is the reason a hybrid approach might be the golden middle ground where some static analysis is done together with on-line measurements. The complexity of the analysis will be reduced some as static analysis methods are well-established, but with the trade-off that the estimates may be more pessimistic than with a pure measurement-based approach. While hybrid approaches might not bring the WCET estimates as far down to the actual WCET of a task, they will to a greater extent be able to ensure safe estimates while still be less pessimistic than pure static analysis.

According to [2] there is still a bit of research left for being ready to use trustworthy probabilistic WCET analysis methods with both static and measurement-based methods. Deterministic measurement methods have been useful although lacking some of the trustworthiness of static methods but might cost the system too much if excessive testing is necessary for good estimates. If further research and development can assure trustworthy results, probabilistic hybrid analysis methods can bring the cost down and be applicable for arbitrarily complex systems.

## 4.4 Fail-safe real-time systems

### 4.4.1 Hardware specifications as domain knowledge

Fail-safe systems have as a goal to quickly put a system to a safe state when given constraints are met. This is a good tool for when it is desirable to use COTS products with real-time constraints, they do not have in the first place. In chapter 2.2.1 the "fail-awareness" method is described where the transition into a fail-safe state is based on the amount of deadline

misses where hardware clocks are utilized to check timing constraints. Using deadline misses makes it possible to use well-established methods for soft real-time programming which may seem easy to implement. In the case of COTS product, this approach cannot be done without the use of hardware timers, and use of domain-knowledge is arguable already implemented for this scheduling. When knowledge about the hardware of a system already is necessary the possibility to utilize this information is more available and can form the base of domain knowledge for the system. The domain knowledge, the hardware information, can be an optional decision-maker for when a transition into a fail-safe approach is necessary.

### 4.4.2 Fail-safe planning based on deadline miss ratio

When deadline misses are allowed, the system is not considered hard, and tasks not considered critical. Without critical tasks, real-time programming can be allowed to be more flexible for adapting techniques more suited for the system. It is in general considered a good idea to let the transition into fail-safe be dependent on a wanted deadline-miss ratio as too many deadline misses normally mean incorrect behavior [33]. But this does not necessarily mean that a deadline-miss ratio constraint corresponds to the need for the system to enter a fail-safe state.

There might be systems where low CPU power is desirable due to limitations in power consumption and heat generation, which is the case for many embedded systems. To get an operational soft real-time system to run with limited CPU power it might be a possibility to allow a high deadline-miss ratio, while still operating with a fail-safe state. The transition to fail-safe could be decoupled from the deadline miss-ratio to depend on constraints based on domain knowledge. By this, a system could spare needed CPU power with less strict timing constraints but still operate with a functioning safe state.

On the other way around, domain knowledge can offer safer systems while using soft real-time systems. The normal fail-safe constraints apply, for example, with the deadline-miss ratio, and in addition, we check out the domain to check if something should trigger a fail-safe state. In this case, the domain knowledge will only offer stricter constraints for the system and offer more opportunities to transition into the fail-safe state. For example, when there is so much wind that a drone is not able to fly forwards, the reference pace is unreachable and a transition into a fail-safe state is necessary. This transition could be solved by executing a predetermined plan, which will now be presented.

### 4.4.3 Fail-safe motion planning

Section 2.2.2 describes an approach for redundant fail-safe planning [32]. There is no description of how such a failure is detected or how to decide when a failure is fatal enough to trigger this action. This is a case where system knowledge can be, and in some systems probably already is, be used. The use of domain knowledge can be used as a decision-maker to decide if the system should enter a safe, or at least a predetermined, state. There is a significant difference between entering a fail-safe state and a mode of operation, but both can be considered different states. A fail-safe state might for example, be to go to a stand-still for a vehicle or land a drone, while a mode of operation could be to slow down a drone. Let us take the drone from section 4.2.5, lowering its speed makes for slower

dynamics for its control system, and can give a job like image processing onboard better time to find the power-line in areas where it is harder to distinguish the power line from the terrain. The lowered speed might be necessary due to more wind, if the wind got so strong that it would have trouble following course, a fail-safe plan could be executed to make the drone land safely on the ground. This is an example of how domain knowledge can be used to make such plans and be a part of the decision to enter a fail-safe state.

## 4.5 Systems with dynamic deployment strategies

Dynamic deployment is in this thesis defined as a collecting term for strategies that use some sort of online decision-making as a part of their real-time scheduling. This offers many exciting strategies that alternate a lot from classical approaches. Although it may not be a very well-defined definition, there are some common lines for many of these types of strategies. They do not solely rely on sufficiency and deadlines, but rather makes use of more information to plan out decisions. This information may very well be domain knowledge and some suggestions will be made in this section based on the dynamic deployment methods described in the literature review in section 2.3.

### 4.5.1 Power-constraint systems

In section 2.3.2 a task set is chosen dynamically during online-execution based on a comparison between offline analyzed utilization and online measured utilization. What makes this strategy so smart is that it does not actually need any additional information than what a general real-time system needs as it only uses time constraints to make its decisions. Assuming tasks in the system have different execution times available, the strategy simply makes room for the possibility that execution may lead to better performance than the offline analysis suggests. As the main problem with the offline analysis is pessimism, this is both a realistic outcome during execution and a smart way to use already well-established strategies to its own advantage. These power-constraint systems do benefit a lot by letting a task execute for a longer time than it actually needs to lower its power consumption.

As the offline analysis still needs to be done and timing constraints will need to be met, these strategies do not offer any way to avoid over-dimensioning of CPU-size and power but it works great for the purpose of lowering the power consumption of the CPU. To extend the execution time without changing its period is nothing but a harming move in real-time scheduling theory. The utilization becomes larger, it is in general harder to ensure timing constraints and mathematically it makes no sense when smaller execution time is available. But as domain knowledge is utilized and it is known that power consumption, rather than time efficiency, is the pressing constraint, the system will benefit more where this is considered.

### 4.5.2 Dynamic change of scheduling strategy

A change of scheduling strategy during run-time is an interesting approach when it can be expected that the execution time of tasks will change after some time. This is an assumption that will not normally be done as it would be easier and safer to work with

constant execution times, like the WCET estimate, of a task. But in systems with dynamic domains, it is reasonable to assume that there will be a variable load on the system in different scenarios. When this information about the domain is known, different constraints can be used as a decision-maker for when a different scheduling strategy can be chosen over the current one. From section 2.3.3 there are overload conditions for when to change from one scheduling technique to another. This decision could be done with domain knowledge-based variables instead, or as an option, to overload. To wait for overload of a system is not a favorable scenario and can in some situations cause a critical situation. By making use of other conditions than deadline misses a system may be considered safer and this approach can be applied to all kinds of systems, not only soft real-time.

### 4.5.3 Dynamic deployment of soft real-time systems

By allowing deadline misses for real-time tasks the rigid regime of hard real-time is broken and new opportunities open up for scheduling strategies. There are many ways to approach these soft real-time systems which benefit from dynamic decisions and optional conditions. Although soft real-time scheduling is also mostly approached with general strategies, the less rigid shape of them can make room for domain knowledge to be a part of the strategy.

From section 2.3.4 a soft real-time system is described where a task may take any period inside a given interval based on weighting and method used to approach the LP problem. Periods for the tasks are chosen in a way to keep the total utilization below the overload condition with different methods. The methods suggested for choosing periods are general and smart, but the overload condition is assumed and not discussed further. While one can set a general overload condition, domain knowledge can offer what a specific system can handle of overload and give the condition that can minimize CPU utilization. This means that this dynamic deployment approach offers a general method for soft real-time systems, while domain knowledge can decide how to make this method as efficient as possible for a given system.

## 4.6 Applying domain knowledge for a proof-of-concept

A proof-of-concept was done to show how domain knowledge can be implemented as a part of the system where it normally would not. The field of control systems was chosen because their dynamic nature might be the most vulnerable to pessimistic execution time analysis. In the next chapter, an implementation of a mode-based control system for an autonomous ship will be shown. Sampling frequency and other discrete update frequencies are normally held constant in a system, but as mentioned in section 3.1 can periodic updates make for a natural set of real-time tasks. Making it mode-based makes simulations possible without too much complexity and makes it possible to implement on already existing systems.

Chapter 5

# Mode based controller system with domain knowledge

This chapter will explore a specific example where the use of domain knowledge offers a new way of scheduling a real-time system. The example is based on autopilot control of a marine ship and will use control theory as well as linear system theory to build a model to work with[1].

## 5.1 System description

### 5.1.1 Equations of motion for the ship

The ship is modelled by what is called a low speed maneuvering model for dynamic positioning. This can be done by assuming the marine vehicle moves in a slow-paced motion, which is reasonable for larger marine vehicles such as transport ships [21].

$$\dot{\boldsymbol{\eta}} = \boldsymbol{R}(\psi)\boldsymbol{\nu} \tag{5.1}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} + \boldsymbol{w} \tag{5.2}$$

where

- $\boldsymbol{M}$ - System inertia matrix
- $\boldsymbol{C}$ - Coriolis-centripetal matrix
- $\boldsymbol{D}$ - Damping matrix
- $\boldsymbol{\tau}$ - Vector of control inputs

[1] The system is based on an assignment given in the subject of Linear System Theory[1]. Answers and Matlab code are heavily inspired by the work on a previously made lab report by the author [38], and a well-made and structured Github repository of previous students [27]

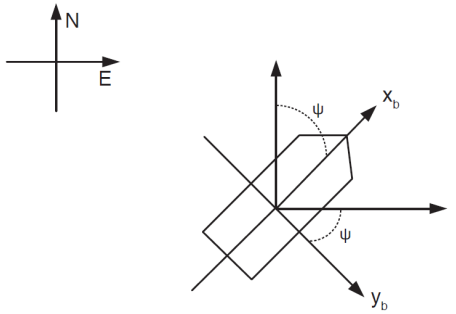
- $\boldsymbol{w}$ - Vector of environmental disturbances, in this example these are due to waves and current
- $\boldsymbol{\eta}$ - NED positions [x, y, $\psi$]. x is the northern position of the ship, y is the eastern position of the ship, $\psi$ is the angle between the x-axis and the vector that points in the direction the ship is facing.
- $\boldsymbol{\nu}$ - BODY velocities [u, v, r]. u is velocity in x-direction, v is velocity in y-direction and r is rotation velocity about the z-axis.

This system will be further simplified by assuming a constant movement speed, $u = u_0$, and assuming the system will only experience small changes for the heading angle $\psi$. Since the pace is assumed low, the non-linear terms in the matrices $\boldsymbol{D}$ and $\boldsymbol{C}$ are negligible and is treated as constant matrices using the movement speed $u_0$. The only control input will be to change the rudder angle, $\delta$, and the control input matrix can be written as $\tau = B\delta$, where B is a constant matrix. The final model for the ship then becomes:

$$\dot{\boldsymbol{\nu}} = \dot{\psi} = r \tag{5.3}$$

$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{N}(u_0)\boldsymbol{\nu} = \boldsymbol{B}\delta + \boldsymbol{w}_{waves} + \boldsymbol{w}_{current} \tag{5.4}$$

**Figure 5.1** BODY and NED reference frames.




## 5.1.2 Disturbances

To be able to replicate a somewhat realistic system, there will be added measurement noise and disturbances from both current and waves on the ship. The waves are high-frequent disturbances which are modeled as a damped harmonic oscillator. The current is a slowly varying disturbance which will be assumed to only affect the rudder angle with a bias, $b$.

- $\psi$ = average heading without disturbances
- $\psi_w$ = high-frequent component due to the wave disturbance
- $\dot{\xi}_w = \psi_w$
- $b$ = bias to the rudder angle

The final equation of motion for the entire system, with ship and disturbances, can be summarized in these equations:

$$\dot{\xi}_w = \psi_w \tag{5.5}$$

$$\dot{\psi}_w = -\omega_0^2 \xi_w - 2\lambda\omega_0\psi_w + K_w\omega_w \tag{5.6}$$

$$\dot{\psi} = r \tag{5.7}$$

$$\dot{r} = -\frac{1}{T}r + \frac{K}{T}(\delta - b) \tag{5.8}$$

$$\dot{b} = w_b \tag{5.9}$$

$$y = \psi + \psi_w + v \tag{5.10}$$

Where $y$ is the measured heading from the compass. $w_b$, $w_w$ and $v$ are Guassian white noise processes.

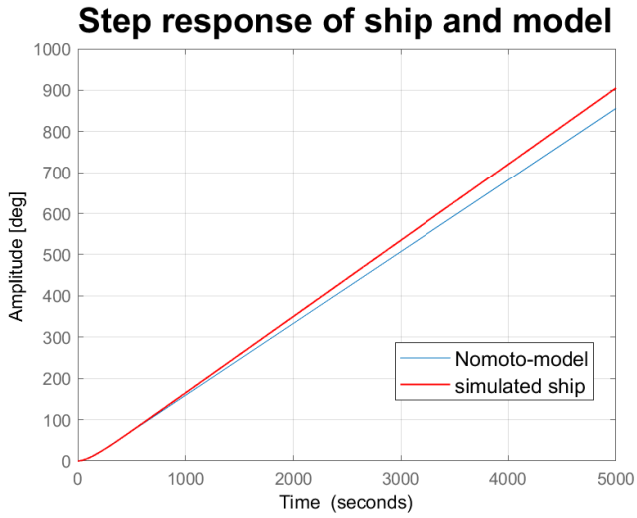
### 5.1.3 First order Nomoto model and controller design

Ships are often using direct measurements of its yaw angle, $\psi$, using a gyrocompass or two GNSS antennas on the same receiver. With direct measurements of the yaw angle, a standardized PD-controller can easily be designed for the system[21]. For such a system, the use of Nomoto modeling of first order is commonly used strategy, which is implemented in this system in the equations 5.7 - 5.8 with time and gain constants, T and K. The corresponding transfer function for the system from the yaw angle to the rudder deflection is:

$$\frac{\psi}{\delta}(s) = H_u(s) = \frac{K}{s(Ts+1)} \tag{5.11}$$

By looking at the step-response of the Nomoto model, using a step input of 1 degree, compared to the simulated ship it can be seen in figure 5.2 that this model is sufficient for shorter periods of time. The inaccuracy becomes significant only after about 2000 seconds, which is acceptable as it is very unlikely that a rudder angle will be held constant for that long.

**Figure 5.2** step-response of simulated ship and transfer function of the Nomoto model




The system will be controlled by a standard limited PD-controller:

$$H_{pd}(s) = K_{pd}\frac{1+T_d s}{1+T_f s} \tag{5.12}$$

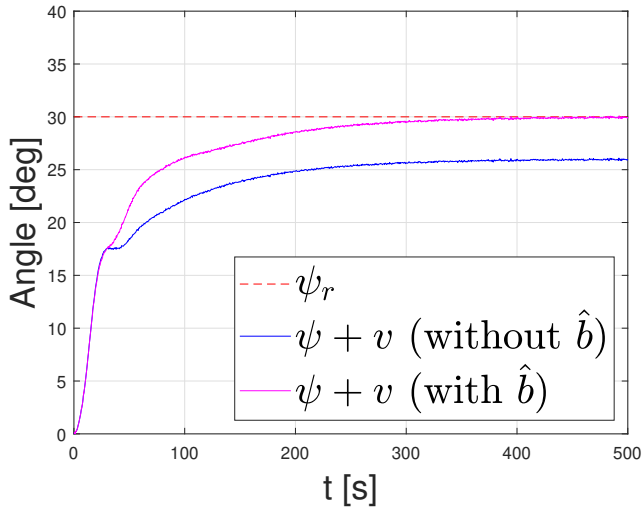
The final system transfer function is obtained by multiplying the system transfer function and the controller transfer function. Since the constants of the PD controller is in the hand of the user to choose, its time constant, $T_d$, is chosen such that the time constant from the system is cancelled, $T_d = T$. The final transfer function for the system becomes:

$$h_0 = H_{pd}(s)H_u(s) = \frac{KK_{pd}}{(1+T_f s)s} \tag{5.13}$$

### 5.1.4 Discrete Kalman filter

A well-known strategy to cope with disturbances is to use a Kalman filter, **KF**. The KF makes an estimate of the measured compass course which makes disturbances due to high-frequent components, such as the waves and measurement noise in this system, less prominent. How good this estimate works depends on sampling frequency, and the effect of changing the frequency of updates will be explored in the next section. In fact, the KF will make an estimate for every state variable in the system, among them the bias $b$. This is useful in this particular system as a PD-conttroller will always suffer due to a constant error after reaching steady-state behavior. By adding the estimated bias as a feed forward component to the ship this steady-state error gets canceled out and the system can reach its wanted reference course. In this system, the KF will be updated with the same frequency variable as the sampling frequency.

**Figure 5.3** The compass course of the ship with and without feed-forward bias from the KF. Compass reference $\psi_r = 30$ and measured compass course $\psi + v$.

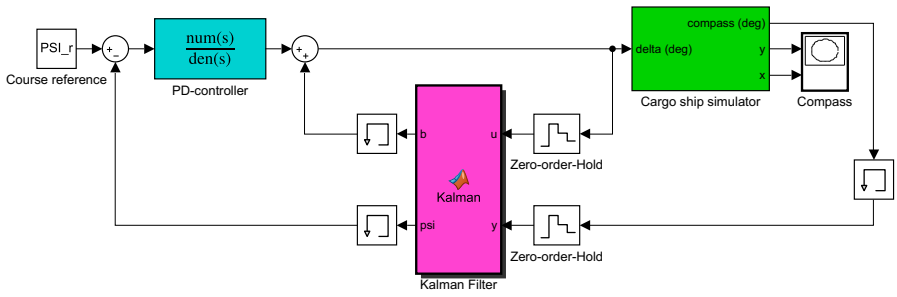



As seen in section 3.3 the KF has a series of computations done at every time step. This is where all state variable estimates get updated and a priori variables for the next calculation get set. How often this is updated will have a major impact on the total system because even if the PD-conttroller is considered a continuous controller, it will only get new values from the KF when a new calculation is done. The last calculated value will be used until a new one is available, and effectively the update of the PD-conttroller is equal as a zero-order-hold (3.2).

### 5.1.5 System overview

The complete system with KF and PD-controller is illustrated below in figure 5.4. The cargo ship is simulated in the green block and its only input is the course given from the control system. The pink block illustrates the KF which uses the measured compass course of the ship and the controlled course from the controller as input. These are used in each calculation of the KF, which outputs are the estimated values for compass course and bias. The PD-controller, illustrated by the blue box, uses the difference between the reference and the Kalman-estimated compass course as input and attempts to apply the necessary gain for this gap to be as close to zero as possible.

**Figure 5.4** The simplified system written in Simulink




The ZoH blocks make the continuous compass course from the cargo ship to discrete values the KF can work with. To simulate this discrete effect on the system, the memory blocks (circled arrow) holds the discrete outputs from until a new value is calculated. This way the PD-controller and the cargo ship will at any time get a continuous signal from the discrete KF.

## 5.2 Real-time scheduling

### 5.2.1 System description

The control of an autonomous ship is a dynamic process that requires quick and consistent computations to cope with disturbances from the environment, for example waves and current as discussed in this particular system. This makes the use of a real-time system a viable option to ensure correct behavior. In addition to the control system, the ship will also have a collision avoidance implemented using an image processing, **IP**, system (e.g. 4.2.5). IP in dynamic environments, such as on a ship, is a computationally demanding task where the execution time can vary a lot from one situation to another.

Normally a ship like this would treat these two systems individually with separate CPUs. In this example, however, these systems will be assumed to run concurrently as individual real-time tasks using the same CPU. It is assumed that these tasks are periodic with no shared resources and that there are no precedence constraints in the system.

- **Task 1 - Update of KF**. This will be a periodic task where the deadline equals to the frequency the KF is updated with. Realistically this should be a whole set of tasks and not only one, but all these possible tasks will be restricted by this update frequency and not offer further information for this example. Alternatively, this task could be seen as a representative of the total set of tasks relevant to the control system.
- **Task 2 - IP system**. This is a periodic task that represents the computational need for the IP system. Much like task 1, it is not realistic to only operate with one task for

this system. As previously mentioned the computational need in IP highly variable, and to set a tight deadline is a demanding task (see 2.1).

To have one task represent a larger set of tasks in a scheduler is not considered as a general solution, but research has shown methods where this can be assumed [11]. For the analog signal to be correctly read after a discretization process, the sampling theorem 3.2.2 needs to be taken into consideration.

### 5.2.2 Different modes

The two tasks in this theoretical system are merely representatives for more complex systems of real-time tasks, but by delegating them a given proportion of CPU time, there is enough information to build a schedule inside these systems. As the case of building such schedules online is a complex one, these will be predetermined and build offline. To do this, the proportions of CPU time is divided into predetermined modes such that offline demands and deadlines can be used. The dynamic actions of this system are now not to choose a scheduling strategy or calculate WCET for tasks, but to change from one predetermined mode to another.

When a group of tasks is dedicated relatively little CPU it would mean that there are conditions in the system domain that allows this group to perform sufficiently without the CPU power. If these conditions change and there is a need for more CPU power to ensure correct behavior, the scheduler would get this information and change to a mode where this group gets dedicated more CPU.

### 5.2.3 Domain knowledge as decision maker

Now different modes can be chosen for different conditions in the domain of this system. The information available from the domain could tell the scheduler when one system could perform sufficiently with less CPU power than another. Challenging conditions within the domain do not necessarily mean easier conditions for another, which would make for an optimal choice of modes, so when choosing modes compromises must be made. The measured values from the domain will act as decision-makers for the scheduler, and with some conditions reached, the system will change from one mode to another. The concept is adaptable to any dynamic environment where the domain consists of other measurable quantities that do not directly affect the real-time properties. This will now be demonstrated using the system of this chapter.

## 5.3 Implementation on autonomous ship model

The previously explained ship model consist of two systems: the autonomous control system and the IP system. Both systems is scheduled in by the same scheduler and is assumed to run on a single core CPU.

### 5.3.1 Obtainable domain knowledge

To decide which mode to operate in, the following domain information is considered:

- **Wind speed and direction**
- **Waves**
- **Current**
- **Humidity**
- **Downfall**
- **sunlight**

A lot of these measurements can easily be read from commonly used measurement tools aboard ships like an anemometer, a weathervane, a hygrometer, and so on. These parameters are important to decide the condition the systems have to work with. As seen in section 5.1, the control system is affected by wind, waves, and current, although wind is not considered in this system. By looking at measures of these, the scheduler will know if the control system is in a challenging environment or not, and this would play a part in choosing the correct mode to operate in. For the IP system, the sight is the main challenge for how much processing power it needs. The sight can be affected by fog, downfall (rain, snow, hail), and sunlight, which are all measurable.

### 5.3.2 Defining modes

For this example, three modes of operation are defined:

- **Mode 1** - Control system weighted. When the condition is sufficiently challenging for the control system and sufficiently easy for the IP system.
- **Mode 2** - IP system weighted. When the conditions for the IP system are sufficiently challenging and sufficiently easy for the control system.
- **Mode 3** - Default mode. Balanced weighting. When conditions are sufficiently challenging or sufficiently easy for both systems.

The weights should be relevant to a predetermined, logical, distribution of CPU between the two systems. For example, if the IP system is a system that requires a lot more CPU than the control system, the CPU distribution in default should be in favor of this. The important part is that the different modes open up for a redistribution of CPU power to favor the system requires it more.

Now the variables from the domain knowledge will be used to decide if one, both or none of the systems is in a challenging condition. The variables wind speed $wi$, wave influence $w$ and current influence $c$ will decide the condition for the control system. The variables sunlight $s$, downfall $d$, and humidity $h$, will decide the condition for the IP system. Each variable will be compared to a limit. A high value will indicate a challenging condition and a low value indicates absent of this factor.

Listing 5.1: Pseudo-code for choosing mode of operation based on domain knowledge

```
% Thresholds for domain variables are set
wth = wave influence threshold
with = wind speed threshold
cth = current influence threshold
sth = sun strength treshold
dth = downfall treshold
hth = humidity threshold

mode = mode 3

while(online scheduling phase)

  % Variables gets updated with the latest measurements.
  w = wave influence
  wi = wind speed
  c = current influence
  s = sun strength
  d = downfall
  h = humidity

  if (w  >= wth OR wi >= with OR c >= cth)
    % There is a challenging condition for the control system
    if (s > sth AND d < dth AND h < hth)
      % There is not a challenging condition for the IP system
      mode = mode 1
    else
      % Both systems have a challenging condition
      mode = mode 3

  else if (s <= sth OR d >= dth OR h >= hth)
    % There is a challenging condition for the IP system
    if (w < wth AND wi < with AND c < cth)
      % There is not a challenging condition for the control system
      mode = mode 2
    else
      % Both systems have challenging condition
      mode = mode 3
  else
    % Neither of the systems have a challenging condition
    mode = mode 3
```

### 5.3.3 Approaches for real-time scheduling

Now there are three operational modes for the ship to choose between. To be able to schedule a task set when less CPU is dedicated to the relevant system there are many available approaches.

**Option 1 - Different scheduling strategies.** For example, control of a ship requires a stable course, which on larger ships does not change fast due to natural slow dynamics. Therefore, it is reasonable to use soft real-time strategies for its real-time programming, especially in easy conditions. In this case, there could be used a scheduling strategy that allows more deadline misses, and the same task set can be considered in all the different modes. Then the soft real-time strategy could be to use different allowed deadline miss ratio for the different modes with the method described in 2.3.1. The same method could also use soft real-time strategy for easier conditions and hard real-time strategy for tougher conditions.

**Option 2 - Different task sets.** With this option the system could use the exact same scheduling strategy for each mode, but the tasks itself will be defined with more slack in timing constraints. This could be by defining tasks that do the same job, but with less QoS, for example, doing the IP with lower resolution or lowering the update frequency of the KF. An approach like this using online WCET estimates is suggested in [23] and further explored by [3].

**Option 3 - A mixture of both** Perhaps the most viable option, and the best general solution, would be to consider both of the previously mentioned options. When considering the domain knowledge, it might be easier to find optional tasks to perform the same job. And by combining this with a scheduling strategy that allows more slack, the possibility of reaching a seriously less CPU dependent execution is there, of course when the domain knowledge finds conditions that allow it.

Chapter 6

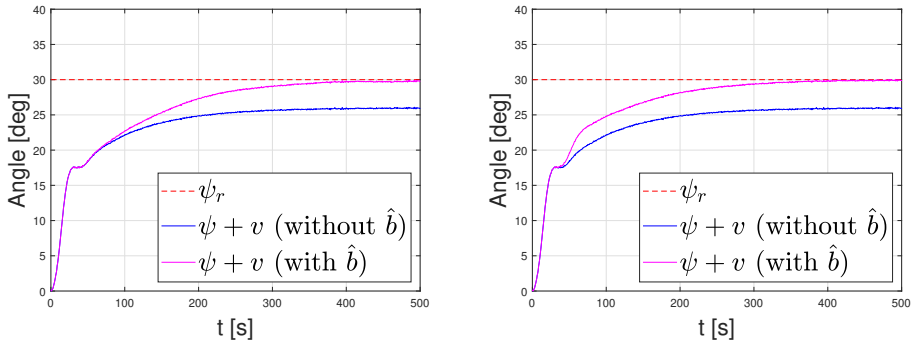
# System response for different sampling frequencies

This chapter will show how different sampling frequencies affect different parts of the control system from the previous chapter. The sampling frequency for discretizing the system is the same as the update frequency of the Kalman filter and will only be referred to as the sampling frequency. In every section, there will be four graphs that shows a response of the full system portrayed in 5.4 with different sampling frequencies of the Kalman Filter. The frequency used is stated below the relevant graph and the response represented by the graphs will be explained at end of every section.

Section 6.1 shows the effect of using feed-forward from the KF estimate of the bias, this feed-forward is implemented in all remaining sections.

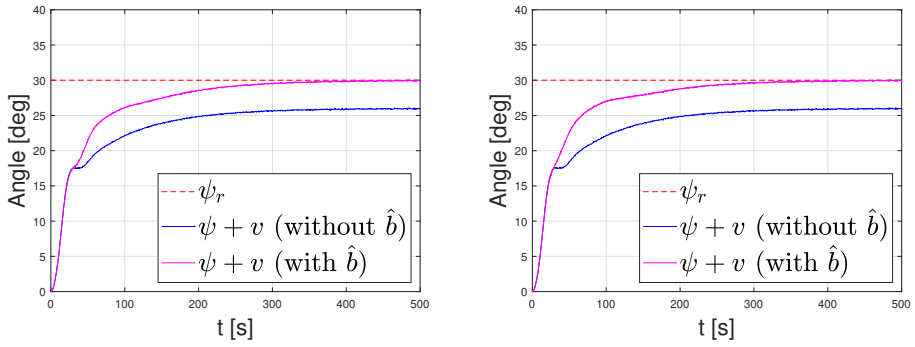
## 6.1 Controller output heading with measured heading

**Figure 6.1** Step-response of controller output with and without KF estimated bias as feed-forward gain.




**(a)** $f_S = 0.1Hz$ **(b)** $f_S = 1Hz$

**Figure 6.2** Step-response of controller output with and without KF estimated bias as feed-forward gain.

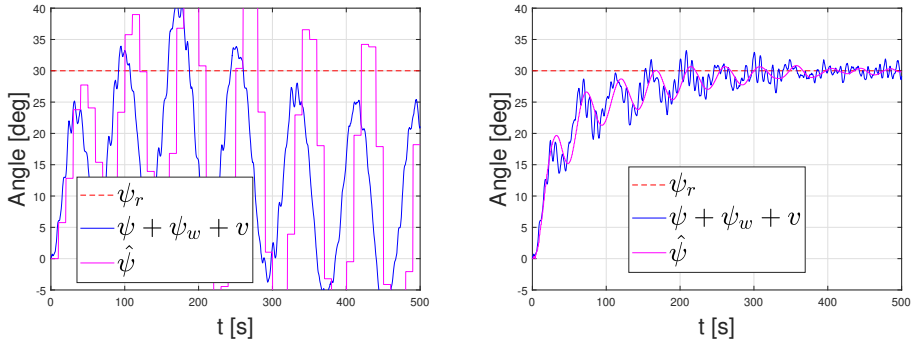



**(a)** $f_S = 10Hz$ **(b)** $f_S = 100Hz$

Figure 6.2 and 6.1 shows the output step-response of the PD controller where the measured heading is used and neither wave or current disturbance are included. The input reference angle of the PD controller is 30 degrees and the effect of adding the estimated bias from the KF, $\hat{b}$, in feed-forward is compared to the response of no feed-forward at all. The different sampling frequencies do not affect the response when no feed-forward from KF is used as no discretization or KF is used. The response when using feed-forward with the estimated bias shows small changes with the different sampling frequencies.

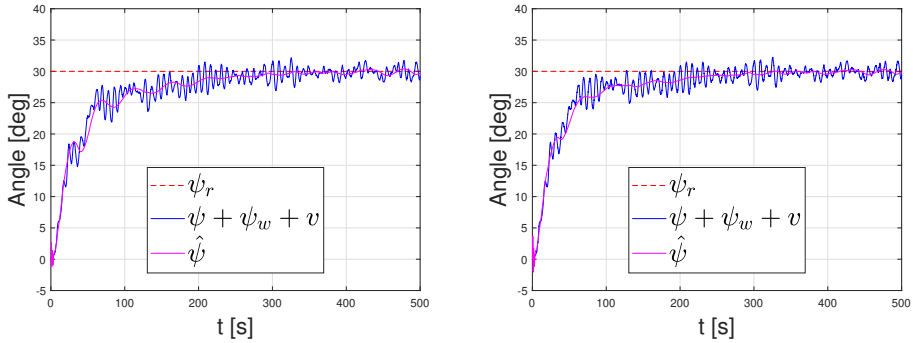
## 6.2 Controller output heading with and and without KF estimated heading

**Figure 6.3** Step-response of controller output with and without KF estimated heading angle




**(a)** $f_S = 0.1Hz$ **(b)** $f_S = 1Hz$

**Figure 6.4** Step-response of controller output with and without KF estimated heading angle.

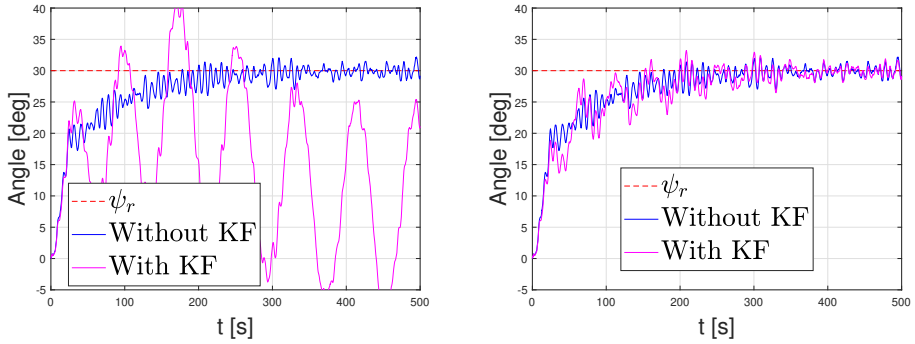



**(a)** $f_S = 10Hz$ **(b)** $f_S = 100Hz$

Figure 6.3 and 6.4 shows the output step-response of the controller when all disturbances are included and compares the response with and without KF estimate of the heading. Figure 6.3a shows that too low sampling frequency gives an unstable output. Remaining figures shows that the KF eliminates high-frequent component due to disturbances, and then spares the physical system for a lot of wear and tear. They also show that a higher sampling frequency better eliminates lower frequent disturbances and gives a smoother, less fluctuating signal.

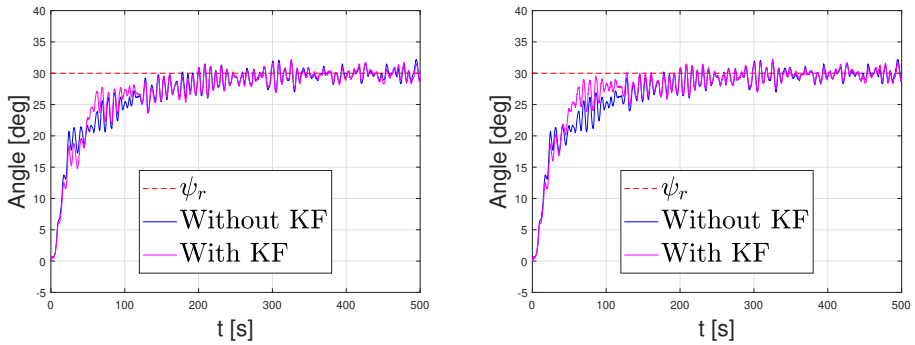
## 6.3 Ship output heading with and without KF estimated heading

**Figure 6.5** Step-response of the output from the physical system with and without KF estimated heading angle.




**(a)** $f_S = 0.1Hz$ **(b)** $f_S = 1Hz$

**Figure 6.6** Comparison between the compass course with and without filtered feedback for different frequency updates of the Kalman filter

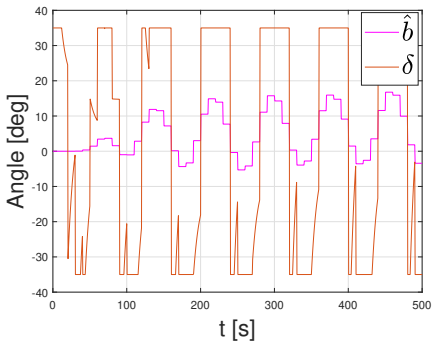



**(a)** $f_S = 10Hz$ **(b)** $f_S = 100Hz$

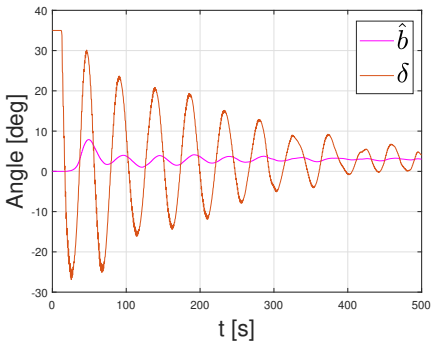
Figure 6.6 and 6.5 shows the response of the physical system. As with the controller output, figure 6.5a shows that too low sampling frequency gives an unstable output. The remaining figures show that higher sampling frequency gives an approximated equal response of the system with and without KF estimated heading angle. This is expected as waves and current will have an effect on the motion of the ship. The benefit of using the KF is in filtering the input signal for the physical system as seen in the previous section.

## 6.4 Rudder input and estimated bias

**Figure 6.7** Step-response of rudder input and estimated bias.

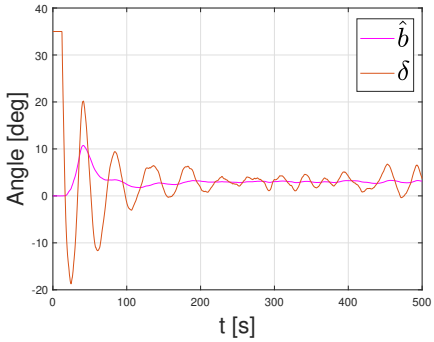



**(a)** $f_S = 0.1Hz$

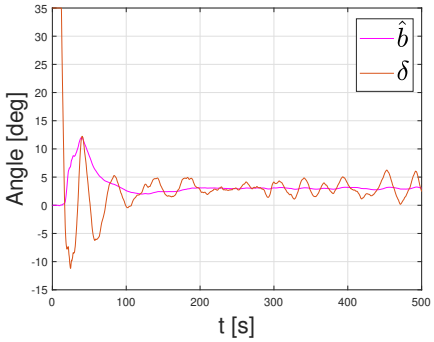



**(b)** $f_S = 1Hz$

**Figure 6.8** Step-response of rudder input and estimated bias.




**(a)** $f_S = 10Hz$

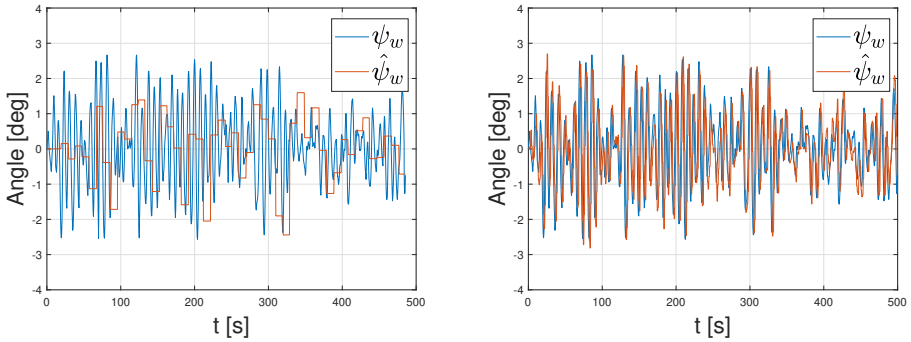



**(b)** $f_S = 100Hz$

Figure 6.7 and 6.8 shows the step response of the estimated rudder input and bias from the KF with different sampling frequencies. Figure 6.7a shows that too low sampling frequency gives unstable output, which explains the unstable responses seen in earlier sections of this chapter. Figure 6.7b shows a stable response as the amplitude is declining, but still with a lot of fluctuation. Figure 6.8 shows that higher frequency gives a quicker and smoother response.

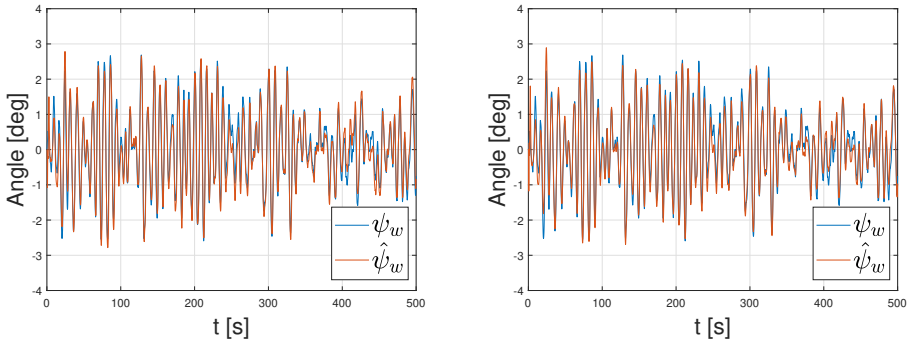
## 6.5 Estimated wave influence

**Figure 6.9** Comparing measured wave influence with KF estimated wave influence




**(a)** $f_S = 0.1Hz$ **(b)** $f_S = 1Hz$

**Figure 6.10** Comparing measured wave influence with KF estimated wave influence




**(a)** $f_S = 10Hz$ **(b)** $f_S = 100Hz$

Figure 6.9 and 6.10 shows how close the KF estimated wave influence is to the measured one. With too low sampling frequency the estimation is very far away from the measurement, but any sampling frequency that gives a stable system gives a very good estimate with only negligible differences.

Chapter 7

# Discussion

In this chapter the main part of the thesis, chapters 4, 5 and 6, will be discussed in terms of use and implementation of domain knowledge in real-time systems. These chapters will be discussed separately with a focus on the possible benefits or downsides with the use of domain knowledge. In the first section, the different categories suggested in chapter 4 will be discussed one by one while the next sections will look at the respective chapters more as a whole.

## 7.1 Domain knowledge categorization

### 7.1.1 Control systems

The biggest benefit pf use of domain knowledge in control systems is to acknowledge how rough the condition of the system's environment is. This information has the potential to decide how much CPU needs to be dedicated to correctly handle the conditions and could spare the system a lot of CPU time, which would be available for other work. It does not, however, solve the problem of overly pessimistic estimations of execution time, and would need just as much CPU power available as without the use of domain knowledge for the worst-case conditions. As these worst cases might be rare, an oversized CPU will have a lot of unused potential and domain knowledge could take advantage of the spare time. This gives systems using controllers a possibility to have more CPU-demanding tasks. Even if the freed-up CPU-time is not used by other tasks the CPU would be less power consuming, which is a great benefit for embedded/smaller systems where this is a huge constraint. In general, the use of domain knowledge in control systems have conceptual benefits and can make room for more efficient utilization of CPU. For bigger systems effects such as power consumption of a CPU might be negligibly small for this to matter and use of multiple CPUs may make the remaining benefits obsolete.

### 7.1.2 WCET analysis

In WCET analysis the motivation of using domain knowledge is clear as pessimistic estimates are a big problem and a lot of real-time systems would benefit a great deal if tighter estimations were possible. This thesis has researched many good approaches for alternative WCET analysis methods where domain knowledge already can be considered in use. The biggest uncertainty of these approaches is the safety of their estimations, which the pessimistic static analysis methods can assure. The alternatives methods must be able to meet the already set standards for safety before they can become a real contender for the static analysis. A lot of research has been done to address this problem and show great potential for safe WCET estimates. This is very much needed as static methods will only turn more pessimistic as complexity grows in programs and the process of finding the longest paths becomes nearly impossible.

Static analysis will simply not be able to keep up with growing complexity. Some deterministic measurement-based WCET analysis has shown to provide safe estimates but costs a lot for a system to execute. Good probabilistic methods will be able to bring the analysis cost down, but it still needs more research for a good implementation as it lacks any industrial use yet.

### 7.1.3 Fail-safe real-time programming

The term domain knowledge could be a variety of things when it comes to fail-safe design as it is big variety between different systems how a transition to fail-safe state is done and why it is needed. Hardware tools are already an established source of information for some fail-safe system and can be seen as domain knowledge. When a system has access to hardware timers for checking timing constraints there are reason to believe that there is additional tools and information available. Often do similar systems offer similar hardware which makes it possible to base fail-safe strategies on a set of given tools. But this assumption is not necessarily true, and it can be hard to make more general solution when available tools are not specified.

Domain knowledge in form of concrete measurable variables offers both additional conditions for when to enter a fail-safe which can be used for both stricter and easier constraints. Whether this is used together with deadline miss ratios or for motion planning for vehicles, this additional information offers nothing but more beneficial behavior. It is how available this information is or how much effort it takes to obtain that is the real bottleneck.

### 7.1.4 Dynamic deployment

When it comes to dynamic deployment strategies there are many good examples of how real-time scheduling can be done differently. From section 4.5.1 it can be shown how power-constraint systems are a great example of how alternative ways of thinking of real-time programming can benefit a system. This use of domain knowledge does not offer a solution to pessimistic task execution estimates nor does it help with making more CPU time available, but it has great benefits for this specific type of system. The domain knowledge is to simply know a priori what to consider when a set of tasks are to be scheduled. Domain

knowledge can offer more opportunities for dynamic scheduling and data to set constraints in already established strategies. It is once again a question of whether it is obtainable naturally from the system or if additional functionality must be added.

## 7.2 Proof-of-concept

The ship model is built with a set of assumptions to make a linear model and simulation of its behavior possible. These simplifications are much used with these kinds of systems and make general solutions possible to use, but there is of course some deviation from reality. The controlling of the ship here is treated continuously with a discrete Kalman filter. In reality, the whole controller system is done in a digital computer with some discretization method which would affect the total CPU usage additionally to the described sampling frequency. The effect of this is predictable and can be accounted for if a fixed time-step method is used but can be a bit more trouble with a variable time-step method. This discretization is unavoidable and the effect of changing the sampling frequency of Kalman filter would be prominent either way.

The proof-of-concept in this thesis has focused on the control system part of this ship, and not much weight has been given to the image processing system. How to implement it or even how increasing or decreasing its dedicated CPU time is not explored in any detail. This proof-of-concept has as a goal to show how a control system could work in synergy with some other task and image processing was chosen as it is a realistic application for autonomous ships.

The three suggested modes in section 5.3.2 gives a good indication of how one can think of mode based controller system. They represent three scenarios that are easy to grasp. The way a mode was entered was suggested in the pseudo-code 5.3.2 where a set of variables was assumed measurable by the system. This was more of an example of how one can decide how tough the environmental condition is for a given system and does not necessarily represent a realistic implementation. A 4th mode could also be considered within the same reasoning by reducing the dedicated CPU power of both systems to the dedications they have when the other is favored. This would happen in the case when both systems are in an easy condition.

## 7.3 Results

By looking at the response of the system it is obvious that implementing a Kalman filter is a great way to deal with frequent disturbances like measurement noise and waves. This is an already well-established conclusion within control theory, and it is more interesting to consider the influence of how frequent the Kalman filter is updated. By looking at the responses with a very low sampling frequency (figures 6.3a, 6.5a, 6.7a and 6.9a), it is clear that the system gets unstable. An unstable response is most likely due to the system not meeting the stability criteria for discretized systems but can also be because the sampling frequency breaches the sampling theorem.

The difference between using 1Hz and 10Hz is significant and shows in general that with lower sampling frequency it takes longer to reach the reference value, as seen by comparing

figure 6.7b and 6.8a. It also gives a more fluctuating response as seen by comparing figure 6.3b with 6.4a and figure 6.5b with 6.6a. But even if the response is a bit slower and more fluctuating, it is possible that the actual effect on the physical system does not cause a very noticeable difference. The high frequencies are still filtered away and the system is still stable with the lower frequency.

By comparing all figures using 10Hz with figures using 100Hz as sampling frequency there are not many big differences. Rudder input and estimated bias (figure 6.8) and ship output heading (figure 6.6) shows no significant difference, and the only true visible benefit from increasing the sampling frequency from 10 to 100Hz is that the controller output is slightly less fluctuating (figure 6.4). The higher frequency does however give overall better response, even if it is not by much, and it is very possible that similar systems have set the sampling frequency as high as the CPU can handle to ensure as good response as possible. If this is the case, it can be interesting to see that frequencies with a difference of 10 times can give very similar results, making the option of lowering the sampling frequency in some situations very possible.

Chapter 8

# Conclusion

By making use of domain knowledge there are multiple possibilities to either free up CPU for other tasks or to lower the CPU power entirely while keeping an acceptable behavior. As this has a good effect on power consumption by the CPU, embedded low-power systems can benefit a great deal from such an approach while bigger systems might find the effect negligible. While one can discuss whether or not this is needed, it is at the very least a step towards a much-needed alternative way of thinking of real-time systems. Many new-thinking scheduling strategies offer well-made methods, but must often assume certain conditions or a given constraint to be used. This is where domain knowledge is needed, this is where decisions can be made for a system based on system-specific information, and this is the argument that although system-specific information is frowned upon it is necessary for a change towards less pessimistic strategies.

What can be considered domain knowledge varies a lot, and the main recurring problem with using it is how available and obtainable it is. Domain knowledge can be something clear like measurable variables in the environment of a control system, or it can be something more abstract like knowing that a system benefits more from having power consumption constraints rather than time. Making a definition of what domain knowledge is and what it can offer does not necessarily mean that new strategies must be made. system-specific information to complement general strategies is seen to be used for many different approaches in many different scenarios, sometimes not even mentioned as a deviation from generality. Domain knowledge is in other words already a part of many used approaches as well as it can offer exciting and smart new solutions.

As the variables of execution time pile up nowadays, as technology grows ever so complex, the off-line analysis only becomes more and more conservative and pessimistic. The conclusion for this work is that more system-specifics can offer more efficient scheduling solution with domain knowledge, while still be categorized to more general fields. Within the field of real-time, there are some natural subcategories where the approach when using domain knowledge is coherent. Although the approaches suggested in this thesis are conceptual and fairly abstract, it is reasonable to believe they would be applicable to systems today, as they mostly make use of already existing information.

# Bibliography


[1] Discrete kalman filter applied to a ship autopilot, 2016. An assignment in the course "TTK4115: Linear System Theory" at NTNU Gløshaugen, Trondheim.

[2] J. Abella, C. Hernandez, E. Quiñones, F. J. Cazorla, P. R. Conmy, M. Azkarate-Askasua, J. Perez, E. Mezzetti, and T. Vardanega. Wcet analysis methods: Pitfalls and challenges on their trustworthiness. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–10. IEEE, 2015.

[3] F. Bakkevig. *Using online worst-case execution time analysis and alternative tasks in real time systems*. PhD thesis, NTNU, 2014.

[4] J. G. Balchen, T. Andresen, and B. A. Foss. Diskret regulering av kontinuerlige systemer. In *Reguleringsteknikk (5. utgave)*, chapter 11, pages 473–517. 2003.

[5] G. Beccari, S. Caselli, and F. Zanichelli. A technique for adaptive scheduling of soft real-time tasks. *Real-Time Systems*, 30(3):187–215, 2005.

[6] G. Bernat, A. Colin, and S. M. Petters. Wcet analysis of probabilistic hard real-time systems. In *23rd IEEE Real-Time Systems Symposium, 2002. RTSS 2002.*, pages 279–288, Dec 2002.

[7] G. Bernat, A. Colin, and S. M. Petters. pwcet: a tool for probabilistic worst-case execution time analysis of real-time systems. In *3rd International Workshop on Worst-Case Execution Time Analysis. WCCET'2003.*, pages 21–38, July 2003.

[8] A. Border. Systems and transfer functions. `https://slideplayer.com/slide/4046828/`, 2020. Slideshow presentation, online accessed 28.05.20.

[9] F. B. Brekke, M. Breivik, and T. A. Johansen. Autosea – sensor fusion and collision avoidance for autonomous surface vehicles. `https://www.ntnu.edu/autosea/`.

[10] G. B. Brown and P. Y. C. Hwang. The discrete kalman filter, state-space modeling, and simulation. In *Introduction to Random Signals and Applied Kalman Filtering*, chapter 5, pages 214 – 225. 3rd edition, 1997.

[11] G. Buttazzo, E. Bini, and Y. Wu. Partitioning real-time applications over multicore reservations. *IEEE Transactions on Industrial Informatics*, 7(2):302–315, 2011.

[12] G. C. Buttazzo. Application design issues. In *Hard Real-Time Computing Systems*, chapter 10, pages 329 – 349. Springer, 2005.

[13] G. C. Buttazzo. *Hard Real-Time Computing Systems*. Springer, 2nd edition, 2005.

[14] P. Castillo-García, L. E. Muñoz Hernandez, and P. G. Gil. Sliding mode control. In *Indoor Navigation Strategies for Aerial Autonomous Systems*, pages 157–179, 2017.

[15] L. de Moura, S. Owre, H. Rueß, J. Rushby, N. Shankar, M. Sorea, and A. Tiwari. SAL 2. In R. Alur and D. Peled, editors, *Computer-Aided Verification, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.

[16] J. Deverge and I. Puaut. Safe measurement-based WCET estimation. In *International Workshop on Worst-Case Execution Time Analysis*. 2007.

[17] R. Dorf, M. Farren, and C. Phillips. Adaptive sampling frequency for sampled-data control systems. *IRE Transactions on Automatic Control*, 7(1):38–47, 1962.

[18] D. Essame, J. Arlat, and D. Powell. Available fail-safe systems. In *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 176–182, Oct 1997.

[19] C. Fetzer and F. Cristian. Fail-awareness: an approach to construct fail-safe applications. In *Proceedings of IEEE 27th International Symposium on Fault Tolerant Computing*, pages 282–291, June 1997.

[20] C. Fetzer and F. Cristian. Fail-awareness: An approach to construct fail-safe systems. *Real-Time Systems*, 24(2):203–238, 2003.

[21] T. I. Fossen. Maneuvering models. In *Handbook Of Marine Craft Hydrodynamics and Motion Control*, chapter 6, pages 163 – 168. John Wiley and Sons, Ltd, 2nd edition, 2020.

[22] W. Hassan. Industrial and embedded computer systems design. Slides from lecture, 2017. from the lecture: Lab lecture 3, A/D Conversion.

[23] S. Hendseth and G. Buttazzo. Exploiting online wcet estimates. *RTSOPS 2012 Final Proceedings*, pages 9–10, 2012.

[24] Hyoung Seok Hong, Sung Deok Cha, Insup Lee, O. Sokolsky, and H. Ural. Data flow testing as model checking. In *25th International Conference on Software Engineering, 2003. Proceedings.*, pages 232–242, May 2003.

[25] W. Kester. Data converter architectures. In *The Data Conversion Handbook*. Analog Devices, Inc., 2005.

[26] R. Kirner, P. Puschner, and I. Wenzel. Measurement-based worst-case execution time analysis using automatic test-data generation. 2004.

[27] P. A. Kjelsvik, M. Bolstad, and M. Kvammen. Github - ttk4115 boat lab repository. `https://github.com/PerKjelsvik/TTK4115-LinSys/tree/master/BoatLab`, 2016. A Github repository containing report and matlab files from the boatlab assignment[1].

[28] K. Kotecha and A. Shah. Adaptive scheduling algorithm for real-time operating system. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 2109–2112. IEEE, 2008.

[29] C. M. Krishna and Y.-H. Lee. Voltage-clock-scaling adaptive scheduling techniques for low power in hard real-time systems. In *Proceedings Sixth IEEE Real-Time Technology and Applications Symposium. RTAS 2000*, pages 156–165. IEEE, 2000.

[30] Z. Li, Y. Liu, R. Walker, R. Hayward, and J. Zhang. Towards automatic power line detection for a uav surveillance system using pulse coupled neural filter and an improved hough transform. *Machine Vision and Applications*, 21(5):677–686, 2010.

[31] C. L. Liu and J. Layland. *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*. The Journal of the ACM, 1973.

[32] S. Magdici and M. Althoff. Fail-safe motion planning of autonomous vehicles. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 452–458. IEEE, 2016.

[33] S. Manolache, P. Eles, and Z. Peng. Task mapping and priority assignment for soft real-time applications under deadline miss ratio constraints. *ACM Transactions on Embedded Computing Systems (TECS)*, 7(2):1–35, 2008.

[34] C. Mazin. Chernobyl, 2019. An HBO tv-series based on the real events behind the Chernobyl incident.

[35] P. Mejia-Alvarez, E. Levner, and D. Mossé. Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(2):284–306, 2004.

[36] J. G. Proakis and M. D. G. Classification of signals. In *Digital Signal Processing*, chapter 1.2, pages 6–11. Simon and Schuster/A Viacom Company, 3rd edition, 1996.

[37] J. J. Shepter. Switch control system. 07 1978. US Patent 4,101,816.

[38] A. Viberg, K. Halvorsen, and J. Kalland. Boat lab assignment, 2017. A lap report of the assignment "Discrete Kalman Filter Applied to a Ship Autopilot" (see [1]).

[39] T. Walter. Characterizing frequency stability: a continuous power-law model with discrete sampling. *IEEE Transactions on Instrumentation and Measurement*, 43(1):69–79, 1994.

[40] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. Measurement-Based Worst-Case Execution Time Analysis. 2005.

[41] I. Wenzel, R. Kirner, B. Rieder, and P. Puschner. Measurement-based timing analysis. In *Leveraging Applications of Formal Methods, Verification andValidation*, pages 430–414. Springer, 2008.

[42] I. Wenzel, B. Rieder, R. Kirner, and P. Puschner. Automatic timing model generation by cfg partitioning and model checking. In *Design, Automation and Test in Europe*, pages 606–611 Vol. 1, March 2005.

[43] M. S. Wiig, K. Y. Pettersen, and T. R. Krogstad. A reactive collision avoidance algorithm for vehicles with underactuated dynamics. 2017.

[44] Wikipedia. Thermistor. `https://en.wikipedia.org/wiki/Thermistor`, 2020. [Online; accessed 26-May-2020].

[45] F. Wolf, R. Ernst, and Wei Ye. Path clustering in software timing analysis. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(6):773–782, Dec 2001.

[46] T. Xu and L. Xu. Bearing signals. `https://www.sciencedirect.com/topics/engineering/bearing-signal`, 2020. Slideshow presentation, online accessed 3.06.20.

[47] K. D. Young, V. I. Utkin, and U. Ozguner. A control engineer's guide to sliding mode control. *IEEE Transactions on Control Systems Technology*, 7(3):328–342, May 1999.