Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 206

A Demonstrator for Networked Control Systems

Daniel Nägele

Course of Study:	Computer Science
Examiner:	Prof. Dr. rer. nat. Dr. h.c. Kurt Rothermel
Supervisor:	Dipl.-Ing. Ben Carabelli
Commenced:	March 1, 2015
Completed:	September 1, 2015
CR-Classification:	C.2.4

Abstract

A rising number of feedback control systems replace their hardwired connections with intermediate networks. These types of control systems are called Networked Control Systems (NCSs) and are often very sensitive to delays in the network, i.e. on their feedback loop. Especially non-uniform delays pose a problem for most controller implementations.
In light of this, research yielded different approaches to stabilize such systems and make them robust. Some of these solutions approach the problem at the controller itself, while others try to mitigate delays and disturbances within the intermediate network. These approaches have one thing in common: Their results are difficult to verify and hardly comparable.
This thesis presents and explains the setup of a demonstrator which can be used to test and compare solutions acting in different intermediate networks using the same, predictable control system. To achieve this, the required theoretical background of control systems is revised and the setup of a inverted pendulum as a NCS demonstrator is explained in detail. Furthermore, different networking setups and their properties are described and documented.

Kurzfassung

Bei einer wachsenden Zahl von Regelkreisen werden festverdrahtete Verbindungen durch zwischengeschaltete Netzwerke ersetzt. Diese Regelkreise werden als Networked Control Systems (NCSs), also als vernetzte Regelkreise, bezeichnet und sind oftmals sehr empfindlich gegenüber Latenzzeiten in diesen Netzwerken, insbesondere bei der Rückführung der Regelgröße. Vor allem ungleichmäßige Verzögerungen stellen die meisten Regelkreise vor Probleme.
Angesichts dieser Schwachstellen existieren verschiedene Ansätze um solche Systeme zu stabilisieren und robuster zu machen. Manche dieser Lösungen versuchen, das Problem mit Hilfe geeigneterer Implementierungen des Regelkreises selbst zu lösen. Andere setzen in den zwischengeschalteten Netzwerken an, sodass diese Verzögerungen und Störungen kompensieren können. Vertreter dieser beiden Ansätze sind jeweils schwer anhand der wissenschaftlichen Arbeiten reproduzierbar und entsprechend kaum miteinander vergleichbar.
Diese Arbeit stellt den Aufbau eines Demonstrators vor, mit dem Lösungen, die in den Netzwerken ansetzen, reproduziert und verglichen werden können. Dementsprechend wird der benötigte theoretische Hintergrund zusammengefasst und der Aufbau eines invertierten Pendels als NCS im Detail erklärt. Desweiteren werden verscheidene Optionen der Vernetzung beleuchtet und deren Eigenschaften dokumentiert.

Table of contents

List of abbreviations

CAN Controller Area Network

CDF Cumulative Distribution Function

CRC Cyclic Redundancy Check

EMAC Ethernet Media Access Controller

MAC Media Access Control

FIFO First In First Out

FPS Frames per Second

GUI Graphical User Interface

HSV Hue-Saturation-Value

IMU Inertial Measurement Unit

IP Internet Protocol

ISR Interrupt Service Routine

L2 OSI Layer 2

LAN Local Area Network

LED Light Emitting Diode

LQR Linear Quadratic Regulator

NCS Networked Control System

ODE Ordinary Differential Equation

PHY physical layer component

PID Proportional Integral Derivative

PS PlayStation

RTS Real Time System

RTT Round Trip Time

SDN Software-defined Networking

SPI Serial Peripheral Interface

UDP User Datagram Protocol

USB Universal Serial Bus

V4L2 Video4Linux2

VPN Virtual Private Network

List of Figures

1. Introduction

Control systems are integral to modern technologies and surround us everywhere. Research in this area has progressed for a long time and various courses of study concern themselves with all aspects of control systems.

Today, control systems are required to be flexible or even distributed while retaining their real-time aspects. Examples of this are their use in wireless sensor networks, remote surgery or in different kinds of autonomous vehicles, such as aerial drones [HNX07].

1.1. Motivation and Definitions

In an introductory book, Control Systems Engineering [Nis00, p. 2], Norman S. Nise defines control systems as follows:

> A control system consists of **subsystems** and **processes** (or **plants**) assembled for the purpose of controlling the outputs of the processes.

While this presents a basic and satisfactory definition in most regards, it does not state how these subsystems are connected. Many control systems must be considered Real Time Systems (RTSs) with hard deadlines, as feedback is required to be virtually without delay [No98].

At the same time, hard-wired connections from sensors to process controllers and their actuators are no longer to be taken for granted. Most often, fixed-delay channels are replaced with less predictable means of communication: Networking hardware has become cheap, wireless sensors are increasingly common, and using those options reduces cost by avoiding additional, dedicated wiring through higher flexibility. These kinds of control systems are sometimes called distributed real-time control systems [No98], or more frequently and recently Networked Control Systems (NCSs) [HNX07].

When decoupling sensors from actuators, their connection quality degrades with increasing distance and a higher number of network segments required to form a link. In addition, the packet based nature of today's networks induces a sampling process between plant and controller, making the NCS a time-discrete system [BA11b]. However, most plants are continuous-time systems and have to be discretized to work in such an environment [BA14].

The solutions proposed for these problems often lack a real implementation and therefore are hard to verify and compare with each other. This thesis presents a demonstrator for NCSs which offers a basic testing environment and is adaptable for any scenario.

1.2. Problems with Networked Control Systems (NCSs)

The network behind a NCS has to satisfy hard deadlines while processing large amounts of data and dealing with other, unrelated traffic. The reasons why NCSs introduce many problems and often only operate with degraded performance have been reviewed in many papers [BPZ00; ZBP01; SQ03; TC03; AS03; Ric03].

All those sources concur, the main problem is the non-uniform distance between samples caused by the following problems present in all networks:

- packet loss
- out-of-order delivery
- collisions between multiple senders

In a packet-based network – whatever the exact specifications are – a trade-off is made between packet size and fragmentation. While fragmentation can mitigate packet loss, it increases problems arising from out-of-order delivery, as most network specifications do not guarantee First In First Out (FIFO) delivery. If the network is shared with other users, maybe even other control systems, cross-traffic or collisions can cause additional delays. Latency is also dependent on network scheduling on the clients and on the intermediate nodes, i.e. if data is sent immediately or after some buffer has been filled.

All of this leads to a non-deterministic distance between samples which either needs to be taken into account by the controller, or should be mitigated by the network to stay within reasonable bounds. As such, most work was done either on controller design or networking protocols [LMT02; GC10].

Multiple solutions exist for both variations [Yan06; HNX07], a popular notion being the use of a Kalman filter or other prediction methods within the controller to compensate variable delays and even the loss of some packets [SFY09; Sin+04].

To mitigate the varying delay, the network needs to replicate the delay-free properties of hard-wired data channels without sacrificing flexibility and scalability. This is, for example, done by using a dedicated network solution, i.e. field-buses like ProfiBus [e.V15] or DeviceNet [Ass15], or by using Software-defined Networking (SDN) (see Section 1.4) and OpenFlow to overcome some limitations of plain Ethernet networks. Their goal is to provide data channels which offer some guarantees regarding performance and latency within a shared best-effort network. Typically, these guarantees depend on the system and its controller and have to be adapted accordingly [Car+14].

1.3. Using the Inverted Pendulum as representative Example

To provide a well-known application which has been a favorite subject of research for a long time [Yam89; PFA05; Zha+05; NAR08; Wan+10], the target of this thesis is to build an inverted pendulum to test and demonstrate solutions and algorithms in the context of NCSs.

The inverted pendulum makes a great example for a control system which mitigates disturbances through a closed feedback loop and is very sensitive to delays when its loop is closed over a network. Using this delay afflicted inverted pendulum as a basic platform, different solutions can be tested in any environment by transmitting the detected angle over an arbitrary network. This can be utilized with a wide variety of different networking specifications, mainly with a simulated Ethernet network controlled by OpenFlow as is implemented later in this thesis.

These problems not only apply to an inverted pendulum, all control systems depending on a slow sensor might still have real-time requirements. As such, not only the example of the inverted pendulum would be a good candidate for studying a NCS.

Any control system which involves a sensor with limited refresh rate can profit from research done on NCS, even if the latency is relatively easy to handle in comparison to a real network.

1.4. Networks for NCSs

To make NCS viable, a possibility is to mitigate any delays affecting the control system. This retains the advantages a NCS has over regular control systems while eliminating some issues the intermediate network involves. Many alternative approaches to mitigate network delays position themselves at the data link layer [e.V15; Ass15]. Consequently, these are not a viable option when using existing networks, as these are typically based on Ethernet.

When dealing with Ethernet (or any packet-based, best-effort network), a popular approach is to use probabilistic control models [Hee+09; BA11a; BA12] to deal with packet loss and latency spikes. However, applying such a model in real communication services present in current networks is difficult and requires a customized routing algorithm, as these models are required to have global knowledge.

Instead it is possible to use state of the art SDN technology [Car+14] which is readily available in much of today's networking hardware. This does not provide a solution on a public network of course, but for example within local networks connected using Virtual Private Networks (VPNs) tunnels which can be integrated into a SDN environment.

In addition, SDN controlled networks can build upon application knowledge which helps immensely when developing a NCS. For example, the expected delays can be communicated with the controller. If, for instance, a critical operation of the controller required above average latency guarantees, a SDN controller might be able to realize these for a period of time by halting other, unrelated traffic.

1.5. Goals of this Thesis

Testing the solutions addressed in the previous section in a real scenario and comparing the results is difficult and often overlooked by the original authors. Most often, the defining criteria for success is the stability of a system, while its overall performance is not considered.

Therefore, this thesis addresses the need to test these solutions by providing a demonstrator for NCS. This is done by describing the hardware setup of a inverted pendulum and determining its exact system model. The software needed to drive the sensors and the actuators will be described and reviewed. To be able to test and compare the different solutions addressed in the previous sections, this demonstrator will support different intermediate networks and is built to be extendable in this regard.

An USB connection with serial communication is the default, but SPI and CAN connections are possible. An Ethernet interface can be used to connect to arbitrary networks, for instance a virtual network powered by Mininet [Lan15]. Mininet integrates well with OpenFlow, making it straightforward to test some of the solutions described in Section 1.4. In addition to these networks, it is easy to add further network interfaces as long as they are supported by the hardware on both ends.

As already noted in Section 1.2, controllers can be adapted to or even built around a

specific network [Yur08]. While this prototype allows some basic controller-side readjustments of the expected delay, it only allows for testing and comparison of solutions acting in or providing the intermediate network with regard to the present controller, a discrete-time state feedback controller with basic prediction (see Section 3.5).

1.6. Outline

The required fundamentals to build and understand (networked) control systems are provided in Section 2. Subsequently, the actual setup of the demonstrator, an inverted pendulum, is detailed in Section 3. This section will deal with modeling the system, specifying the hardware used in the setup, and finally describe the controller used to drive the demonstrator (Section 3.5). After this, different networking aspects will be discussed and evaluated in Section 4, especially introducing the means to test Ethernet networks and SDN. Finally, a general evaluation and a description of future work is offered in Section 5.

2. Background: (Networked) Control Systems

When building a control system from scratch, it is important to do it in a structured way and with a clear understanding of the underlying algorithms. This section attempts to summarize the theory behind control systems and the approach used to build this control system.

2.1. Control Systems

In accordance with the definition of control systems given in the introduction (Section 1.1), every control system employs actuators which are used to control the processes output, and sensors which are used to read input to determine how to drive the actuators to reach the setpoint. In addition, there usually is a disturbance to the plant which can be taken into account by the controller. There are however different kinds of controllers available, each suiting different applications.

2.1.1. Open-Loop Control

Open-loop control systems are the simplest form of control systems. Their input and the disturbances determine their output. They have no sensors to tell if their setpoint has been achieved which makes the correction of disturbances impossible. This kind of control system is easily implemented as a NCS, as increased delays only lead to sluggish controls. A good example is a car's steering wheel, which is exposed to larger disturbances at higher speeds.

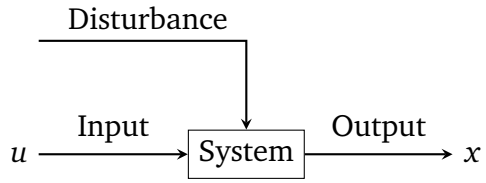


Figure 1: Schematics of an open-loop system.

2.1.2. Feed-Forward Control

Feed-forward systems counteract disturbances when they are detected or anticipated. Like open-loop systems, they do not measure their actuating variable and can not correct their setpoint. A good example is power steering, which counteracts any disturbances, for example those caused by high speeds..

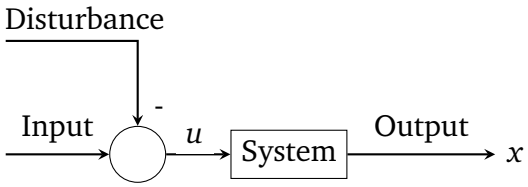


Figure 2: Schematics of a feed-forward system.

2.1.3. Closed-Loop Control

Closed-loop controllers gain feedback by observing the actuated variable and correcting their own output accordingly. This type of control system is suitable to have a control system mitigate disturbances in the system. A good example is a driver steering a car to stay on a path.
This is also the most sensitive type of control system in regards to delays, as the data received on the feedback loop is required to be as current as possible. Therefore, this is the type of control system most relevant to NCS and has to be considered a RTS. This is already apparent from the example given above, as a human driver introduces considerable delay.

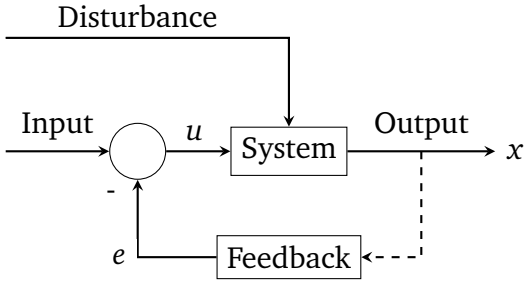


Figure 3: Schematics of an closed-loop system.

2.2. Control Loops and NCSs

As noted above, closed-loop control systems are especially sensitive to delays affecting their feedback loops. This makes them challenging NCS setups, especially when paired with a network of poor quality. NCSs can be defined as follows:

> **Networked Control Systems (NCSs)** are control systems which connect their components, i.e. sensors, actuators and the controller itself, using a network.

This is a concise definition of NCSs in general. However, it does not determine the type of control system used and the specifications of the network. When NCSs are mentioned in this thesis, two things are implied:

- The control system is implementing a feedback loop, i.e. it is a closed-loop system.
- A shared, best-effort, packet-switched network is used to connect its components.

Similar presumptions are made in many research contributions on the topic of NCS [Yur08; Hee+09; Car+14].

2.3. Models of Feedback Control

There are many algorithms for controlling a system which can be assigned to one of two categories [Nis00]. First, there is the classical control theory which operates on frequency-domain functions. A system is represented as algebraic transfer functions which makes it possible to control linear systems modeled as an Ordinary Differential Equation (ODE).

The alternative is to use the modern approach to control theory which uses time-domain functions and results in working with a state space representation. This approach also applies to non-linear systems as it is using general differential equations.

By using vectors for any input, state, or output variables the complexity can be reduced by representing higher order differential equations line by line, yielding a first order equation for each line.

2.3.1. PID Control

A PID Controller (**p**roportional-**i**ntegral-**d**erivative) is one of the most versatile controllers available and a prime example for a classic control mechanism. As shown below (Equation 2.1), the actuated variable u is determined by forming a weighted sum of the error function e, i.e. the offset from the system's setpoint. See Figure 3 from Section 2.1.3 for a visual representation in which the feedback is simplified: The three feedback nodes (with their factors $K_{\{p,i,d\}}$) typically used to depict a Proportional Integral Derivative (PID) controller are simplified and merged into a single node.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \tag{2.1}$$

This is a straight forward approach to control a process, but the parameters are not found easily. $K_{\{p,i,d\}}$ can be set to arbitrary values and have no relation to each other. To determine these deciding properties, either an accurate system model is needed, or – which is the most widely used approach – the controller needs to be tested and tuned thoroughly. Some empirical methods to make finding the controller's parameters manageable are found in scientific literature [WZC00; YZH05; MDF00].

In a NCS, this is even harder, as the non-uniform distance between samples and the delay have to be taken into account when using PID control. To counteract the delay a straightforward and popular method is to interpolate the feedback to reflect the current state using a Kalman filter [Sin+04; SFY09]. Using the singular perturbation technique [Yur08] is a promising alternative.

2.3.2. Control with a Linear Quadratic Regulator (LQR)

The Linear Quadratic Regulator (LQR) is a feedback controller which operates on a state space representation. It yields an optimal controller in reference to a cost function and is applicable to both, continuous and discrete time problems.

For a given state space representation (of the form shown in Equation 2.2) two matrices are defined: Q which represents the cost for errors, i.e. deviations from zero and R which sets the cost of input. They both are part of the following cost function (Equation 2.3), which is optimized by the LQR method.

$$\dot{x}(t) = A \cdot x(t) + B \cdot u(t) \tag{2.2}$$

$$J(u) = \int_0^\infty \left(x^T Q x + u^T R u \right) dt \tag{2.3}$$

With these, a state-feedback control gain matrix K is calculated, for example by using the method introduced by Blind et. al. [BA12]. Calculating this matrix is called the

LQR problem. With this matrix given, the optimal control variable which minimizes the cost is given by the following equation:

$$u(t) = -K \cdot x(t) \tag{2.4}$$

This cost optimization is an algorithmic way of determining a state-feedback controller. However, tuning of the cost matrices Q and R is still required and often presents an iterative process. As before, the delay present in a NCS must be compensated, for example by using a Kalman filter to interpolate the measured state to the present state.

The state space representation required for this controller will be derived within the next section (3.1.2). The approach used in this demonstrator to determine the gain matrix K will be explained in further detail in section 3.5.

3. Building a NCS: The Inverted Pendulum

The obvious choice when constructing an universal demonstrator for NCS is a closed loop feedback system, as it is the most susceptible type of control system when dealing with delays.

The inverted pendulum is probably the most common example object in control systems research and also a closed loop system. As such, it is well defined and its properties are already thoroughly documented [Yam89; No98; PFA05], making it an ideal base for a demonstrator.

The following sections will detail the hardware and software used for this setup.

3.1. Setup and Properties

An inverted pendulum consists of a rod with its pivot point mounted on a cart or carriage (see Figure 4). The carriage needs to be controlled in such a way, that the pendulum's center of mass always stays directly above the pivot point, i.e. that the pendulum does not fall over.

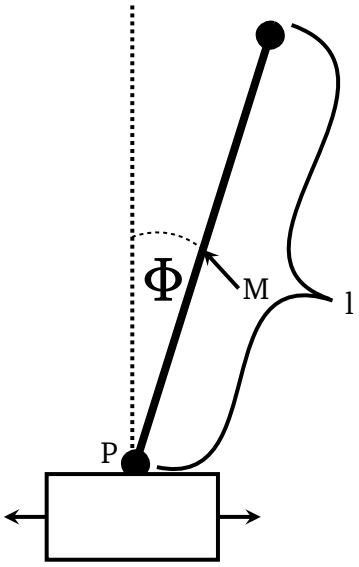


Figure 4: A sketch of an inverted pendulum. M denotes the pendulum's center of mass, l its length. The actuating variable is the pendulum's angle Φ from the vertical.

This demonstrator's inverted pendulum is not implemented with an Inertial Measurement Unit (IMU) like most implementations are, but rather using a consumer-grade camera which detects the pendulum's current inclination. First, this is because of practical reasons: When using cheap hardware – like the carriage of a desktop printer – an IMU mounted on the pivot point might receive too much vibration and measurement noise to be of any use. Second, using a camera adds credibility to the setup as a NCS, as having two micro-controllers sitting side by side without a hard-wired link is not a plausible reason for making it a NCS.

There is also the obstacle of driving the camera: Most controllers and actuators are run using a micro-controller which generally does not offer possibilities to interface with a camera. As seen in Figure 6, this demonstrator circumvents this by placing the camera at a computer which can communicate with the micro-controller used to drive the actuator.

This makes this setup distributed in the sense that the camera and the detection algorithm have no precise frame rate (see Section 3.4.1) and the components interacting with the sensor and the actuator are decoupled.

Having a computer on one end of the transmission channel also has other advantages which will become obvious in Section 4.2. This makes the inverted pendulum a great example for a distributed real-time control system, even in the absence of an actual network.

It does not matter whether sensor data (the angle) or acceleration parameters are transmitted between the computer with the sensor from a networking perspective.

However, locating the controller with the actuator is likely to perform better in presence of packet losses and is proven to be optimal under certain assumptions [RK08]. Therefore, this implementation locates the controller directly at the pendulum, i.e. on the micro-controller.

Following from this decision, the demonstrator in its current form closes the feedback loop over a network in only one place, from sensor to controller. To reflect a fully distributed NCS in a more precise way, an additional component could be added to the setup. The assumption that the controller is nearby at least one of the components and is networked using a fast, dedicated link is however reasonable and serves the example of a camera controlled inverted pendulum very well.

3.1.1. System Model Deduction

To successfully design a control system, the system must be described and modeled. The equations necessary for a full system model need to describe the pendulum's moment of inertia as well as its kinetic and potential energy.

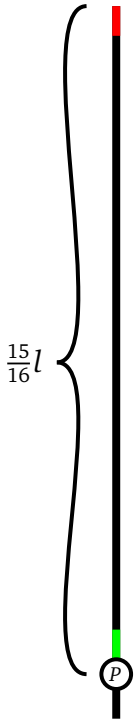


Figure 5: Another schematic view of the pendulum and its pivot point P.

The pendulum's moment of inertia can be simplified if it presents a thin rod without any additional mass mounted on top (like the illustration in Figure 5). The following formula applies for the moment of inertia when the rod is mounted at its center:

$$I_{end} = \frac{1}{12} \cdot m \cdot l^2 \tag{3.1}$$

The pendulum's pivot P is of course not in the middle of the rod, but neither directly at the end of the rod: It is $\frac{1}{16}$ of its full length from the bottom (Figure 5). This means, the moment of inertia of the pendulum is given as a direct result of the Huygens-Steiner theorem. This theorem states that the moment of inertia increases by the mass times the squared distance from the center of mass:

$$I_P = \left(\frac{1}{12} \cdot m \cdot l^2\right) + \frac{7}{16} \cdot m \cdot l^2 = \frac{211}{768} \cdot m \cdot l^2 \tag{3.2}$$

The assumption that no additional mass is mounted on the pendulum also yields the distance from the pivot to the pendulum's center of mass ($d = \overline{PM} + \frac{1}{16}l = \frac{l}{2}$). The following equations and their derivatives define the system's geometry which is necessary to calculate the system's energy. The indices refer to the pendulum's base and center of mass, according to Figure 4, i.e. (x_P, y_P) and (x_M, y_M) are the Cartesian coordinates of the pivot P and the pendulum's center of mass M, respectively.

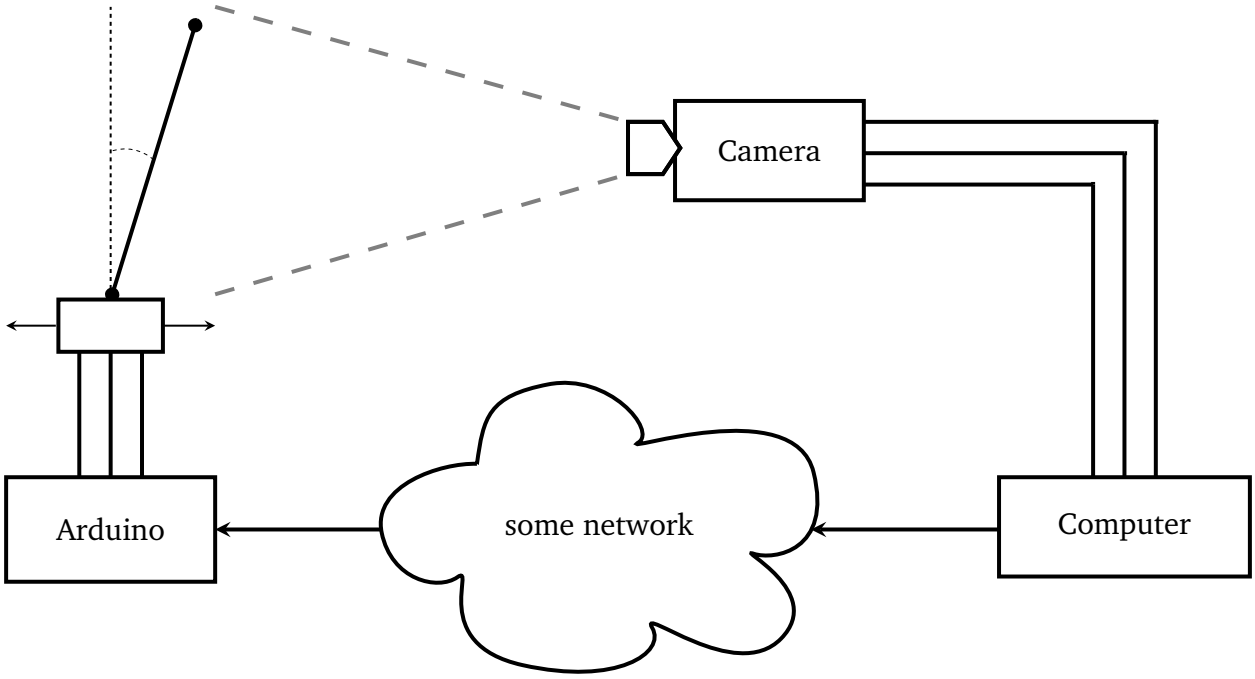


Figure 6: A sketch of the whole setup. Single arrows are used for high latency connections while triple links indicate a hard-wired, low latency connection. The controller with all its components is located on the Arduino, while the computer is driving the sensor, a PS Eye camera. The intermediate network is variable.

$$
\begin{aligned}
x_P &= x & \dot{x}_P &= \dot{x} \\
y_P &= 0 & \dot{y}_P &= 0 \\
x_M &= x - \frac{l}{2} \cdot \sin(\Phi) & \dot{x}_M &= \dot{x} - d\dot{\Phi}\cos\Phi \\
y_M &= \frac{l}{2} \cdot \cos(\Phi) & \dot{y}_M &= -d\dot{\Phi}\sin\Phi
\end{aligned}
$$

With these, the combined velocity of the center of mass M and thus the kinetic energy T as well as the potential energy V are given by the following equations:

$$v_M^2 = \dot{x}_M^2 + \dot{y}_M^2 = \dot{x}_M^2 - d\dot{x}\dot{\Phi}\cos\Phi + d^2\dot{\Phi}^2 \tag{3.3}$$

$$T = \frac{1}{2}mv_M^2 + \frac{1}{2}I_P\dot{\Phi}^2 \tag{3.4}$$

$$V = m \cdot g \cdot y_M \tag{3.5}$$

These yield the Lagrangian ($L = T - V$) which can be solved for $\ddot{\Phi}$ using the Euler-

Lagrange equation. Note that x_p is the sleds position and Φ is the pendulum's angle.

$$\ddot{\Phi} = \underbrace{\frac{d}{d^2 + \frac{1}{3}l^2}}_{=F,\text{ a constant factor}} \left(\ddot{x}_p \cos(\Phi) + g \sin(\Phi)\right) \tag{3.6}$$

The calculation yielding this result can be reviewed in Appendix A. The inverted pendulum behaves approximately linearly for small values of Φ and probably would not be able to recover from greater inclinations considering the commodity hardware used in this setup. Thus, this equation may be linearized around $\dot{\Phi} \equiv \Phi \equiv 0$, as a linear ODE is required for LQR design:

$$\ddot{\Phi} = F \cdot \ddot{x} + F \cdot g \cdot \Phi \tag{3.7}$$

Note that $\ddot{x}$ is the acceleration of the carriage, i.e. the controllers output u, and the constant factor F is defined in Equation 3.6.

3.1.2. State Space Representation

The full state space representation is given by the following equations:

$$\underbrace{\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{\Phi}_1 \\ \dot{\Phi}_2 \end{pmatrix}}_{\dot{x}} = \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & Fg & 0 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \Phi_1 \\ \Phi_2 \end{pmatrix}}_{x} + \underbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \\ F \end{pmatrix}}_{B} \cdot u \tag{3.8}$$

$$\underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \Phi_1 \end{pmatrix}}_{y} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{C} \cdot x \tag{3.9}$$

In these, x is the state vector which is composed of the cart's position x_1, the cart's velocity x_2 $(= \dot{x}_1)$, the pendulum's angle Φ_1 and the pendulum's angular velocity Φ_2 $(= \dot{\Phi}_1)$.

The matrix A is called the system matrix, with B being the input matrix and C being the output matrix. Note the constant factor F present in these matrices which is defined as shown in Equation 3.6. The most important parameter is u, the control variable which represents the acceleration the actuator needs to achieve.

These equations are still in their continuous-time form and need to be discretized as follows:

$$x_{k+1} = A_d x_k + B_d u_k \tag{3.10}$$

$$y_k = C x_k \tag{3.11}$$

Obtaining the discretized matrices A_d and B_d is shown in previous research [BA12], but summarized in Appendix B.

3.2. Actuator: A modified Desktop Printer's Carriage

A goal of this thesis was to built a demonstrator with commodity hardware at hand. In this case, the solution is to reuse the carriage of an old desktop inkjet printer and modify it to fit the application.

3.2.1. Setup and Hardware Properties

To accommodate the pendulum, the printing head was removed and the carriage was fitted with a small aluminium profile serving as a pivot for the pendulum. No other modifications have been made. The carriage can move on a 0.38m segment, the pendulum is a simple metal rod 4mm in diameter and 0.265m in length.

A desktop printer's carriage is typically driven by a stepper motor attached to a belt. This stepper motor provides sufficient acceleration and speed to compensate smaller deviations from the setpoint. Its top acceleration is at least $1\frac{m}{s^2}$ and the top speed is at least $1\frac{m}{s}$, both of which are more than required to balance the pendulum (see Section 3.5.2). The whole length of the carriage is addressable by 950 full steps (1.8° each). These values are lower bounds which have been determined experimentally, as the actual values depend on the physical load and the gears within the motor.

3.2.2. Actuation with a Micro-controller

The stepper motor is actuated using an A4988 Micro-stepping Driver [All09] and an Arduino Due [LLC15]. The controllers output, i.e. the acceleration needed for stabilizing the pendulum, is converted to a time delay for which the controller needs to wait before the next actuation of the stepper motor. This conversion is done based on the following well known equation.

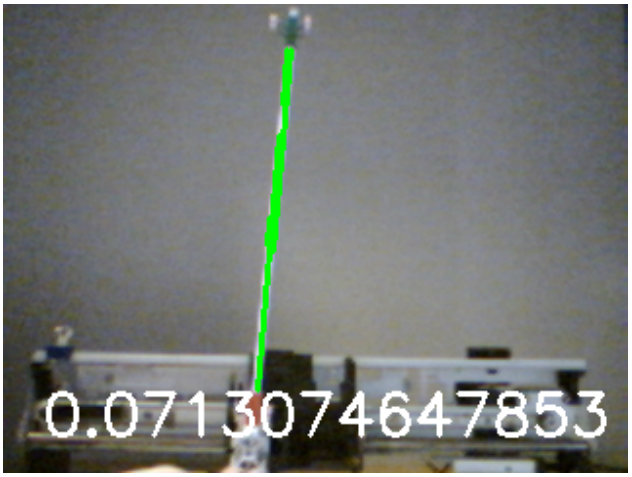
$$s = v \cdot t + \frac{1}{2} a \cdot t^2 \tag{3.12}$$

By substituting s (in meter $[m]$) with the known base stepping angle δ (and thus, converting all other units from meter to radians), the formula can be solved for t, the interval between actuations which defines the steps done by the motor. The angle δ is typically $\frac{1}{8}$ by default and can be set to different values using the A4988 [All09].

$$t_{1,2} = \frac{-v \pm \sqrt{v^2 \pm 2 \cdot a \cdot \delta}}{a} \tag{3.13}$$

Only the smaller, positive t is a relevant solution. To find the correct one, some simple cases have to be distinguished. Both plus-minus signs always become the same operator and when accelerating further in the current direction of movement, both are set to the current direction.

When changing directions, i.e. decelerating and then accelerating in the other direction, the discriminant's sign has to be observed. If it is greater or equal zero, the cart still has to decelerate before changing directions, i.e. the same plus-minus signs are used, but the intervals grow gradually. If the discriminant reaches into the negative, the plus-minus signs (and thus, the direction) are reversed and the default mode of keeping the current speed or accelerating further is restored.



(a) **Detection using Hough Transform.** The angle is calculated in regard to the detected horizon line which is not shown as an overlay.

(b) **Detection using only two of three possible markers.** The offset between the actual horizon line and the debug output is clearly visible.

Figure 7: Example GUI output for both implemented algorithms (3.3.3 and 3.3.4).

3.3. Sensor: A PS Eye Cam for Visual Angle Detection

In addition to a household printer carriage for the actuator, a standard PlayStation (PS) Eye Cam was used as a sensor. This camera is not only cheap, but also has some convenient properties for computer vision.

Other consumer grade options considered for this setup include using a Raspberry Pi Camera [Pic] and repurposing the Leap Motion controller [Lea]. The Raspberry Pi Camera provides a much better resolution but offers only a poor frame rate. Contrarily the Leap Motion controller offers a much better frame rate, but also has high cost and a difficult driver situation (no Video4Linux2 (V4L2) support yet).

3.3.1. Hardware Properties

The PS Eye was chosen for its good price-performance ratio. It delivers frames with a 320px to 240px ratio at a theoretical maximum of 187Hz [How14], although the final implementation uses a mere 125Hz. The PS Eye Cam also features a fixed focus lens, with a field of view of either 56°or 75°. The absence of auto focus also improves the stability of computer vision algorithms which is reflected by its popularity for computer vision projects [Kir09].

3.3.2. Setup

The PS Eye is operated with a Linux computer which displays a Graphical User Interface (GUI) showing a live feed with some debugging output (see Figure 7) and a calibration window. The operator needs to use this GUI to select appropriate Hue-Saturation-Value (HSV) color values specific to the elements the algorithm is trying to detect. The HSV color space is used to make this manual calibration intuitive and straightforward. The detected angle and the recognized geometry are shown as an overlay in the live feed window.

The live feed is also used to position the camera correctly to ensure that the view is high and wide enough to accommodate the pendulum and the width of the carriage. In addition, the operator can easily detect problems with the lighting condition or with an irregular background. To prevent angular distortion by having the camera set up at a skewed point of view, the operator's (likely inaccurate) efforts are assisted by the detection algorithm.

To aid computer vision algorithms, some preparation of the raw frames is required. Every frame is converted to the HSV color space to simplify comparison with the calibrated values. Using this comparison, the filtered image data becomes a binary representation of the current frame. These binary pictures are available to the operator for easy online tuning of the values.

To evaluate the performance and practicability of different methods, two algorithms were implemented both of which can be applied on these binary images. All of this sensor framework make use of the OpenCV [Gar15] library and its Python [Ros15] bindings.

3.3.3. Detecting Angles using Hough Transform

The first method implemented for measuring the pendulum's angle is using the Hough Transform [Hou62] for detecting the lines formed by the pendulum and the carriage, the second of which serves to solve the problem of a skewed point of view. The operator needs to calibrate two HSV values which should reflect the pendulum's color and the color of a marker on the horizontal through the pendulums pivot. Obviously, both need to have an unique coloring and need to stand out in the cameras view.

The detected lines always consist of multiple lines with nearly the same angle which makes using the average of their angles necessary. The angle is calculated using standard vector arithmetic in respect to the horizontal determined using the detected colored marker.

3.3.4. Detecting Angles using Markers

As described before, markers require calibration by the operator who has to choose HSV values for each one. This algorithm uses one marker at the pivot on the carriage, one at the tip of the pendulum and optionally another on the horizontal line from the pivot marker. The first two are visible as green and red markers in Figure 5.

The last marker is not strictly necessary, but it helps reducing measurement offsets induced by a skewed camera setup as shown in Figure 7b. When the camera is mounted precisely, this third marker can be left out as it accounts for a third of the processing time which may be critical depending on the computer used.

The algorithm calculates the image moments [Hu62] of the binary representations which are used to find the center of mass of the largest cluster that matches the calibrated values. This leads to precise and stable marker locations which are robust against fluctuations in lighting conditions and also dampen the measurement noise.

Using these locations, i.e. three points known by their coordinates, the angle between the pendulum and the horizontal can be determined using basic vector arithmetic. Some details of the code are found in Appendix C.

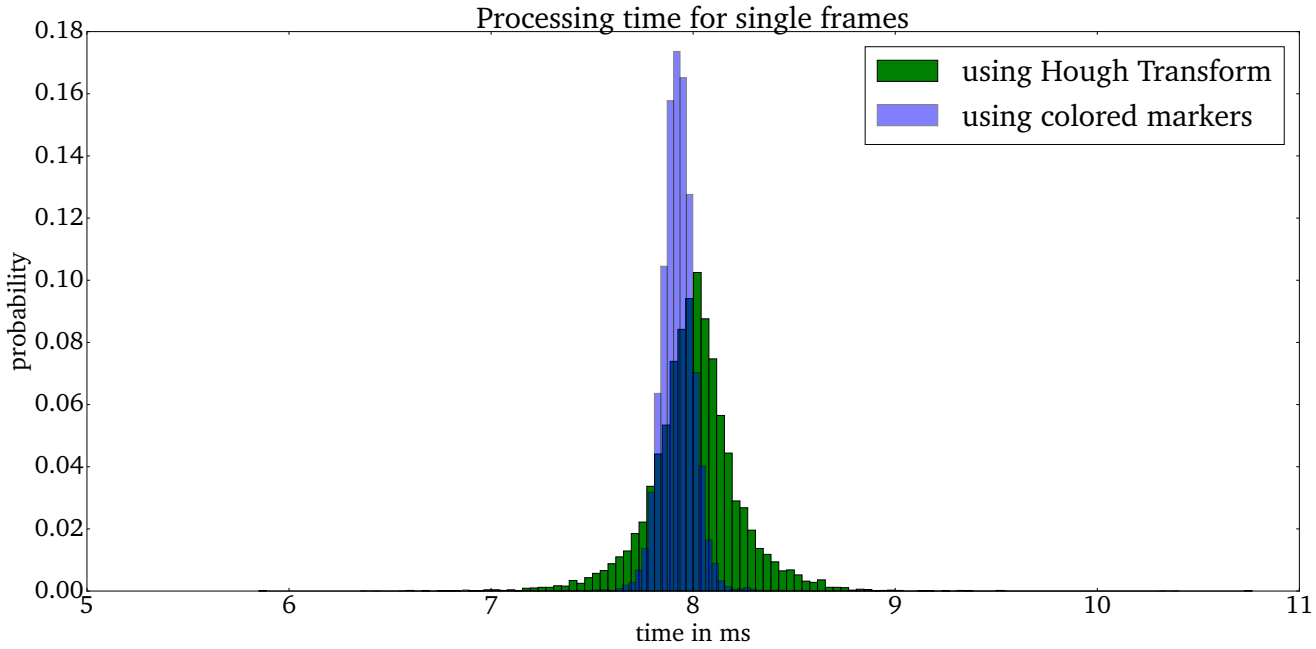


Figure 8: Comparison of processing times in 125Hz mode for single frames when using Hough Transform and colored markers. Hough Transform shows a higher variance (0.05038ms, as compared to 0.01093ms) at the same quality of results. Hough Transform is also slightly slower on average (8.00436ms, as opposed to 7.92380ms).

3.3.5. Angle Detection Implementation

As shown in Figure 8, the approach using colored markers (Section 3.3.4) yielded a lower variance of output delay than the other algorithm (Section 3.3.3). While this variance was the deciding factor when choosing the algorithm to use, the quality of the results is also slightly better when using colored markers instead of the Hough Transform. Figure 9, which was generated using a static angle, shows the jitter in results for both methods of measurement. The algorithm using colored markers provides the cleaner results but the introduced jitter still is of the same magnitude.

The maxima in the blue histogram (colored markers) of Figure 9 are due to the low resolution the camera takes samples with and occur at the arithmetically possible locations. The algorithm using Hough Transform does not group its results into buckets, as an average is used for the final value.

Especially because the delay is less predictable, a deciding factor in NCSs, the sensor built around the Hough Transform was not chosen for further optimization. The final implementations performance is reviewed in the next section.

3.4. Signal Path

Even though the setup described above does not yet feature a network, information about the angle arrives at the controller with a considerable delay. The reasons for this are mainly two bottlenecks:

- The sensor itself.

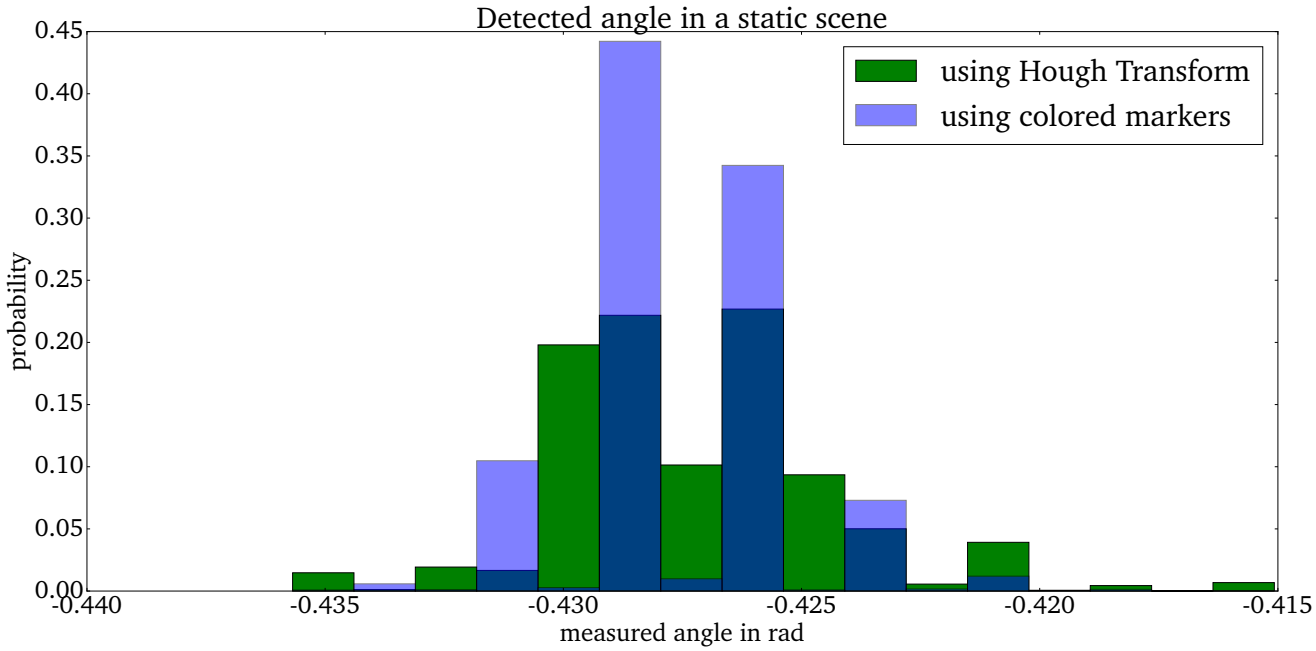


Figure 9: When using colored Markers, the jitter of the detected angle was $4.04510 \cdot 10^{-6} rad$, the mean angle detected was $-0.42742 rad$. When using Hough Transform, the jitter was $8.34698 \cdot 10^{-6} rad$, the mean angle detected was $-0.42726 rad$.

- The transmission times to the controller and to the actuator.

3.4.1. Latency at the Sensor

The delay caused by the sensor is mainly dependent on the Frames per Second (FPS) available from the camera. Driven at 125Hz, the average distance between frames is 8ms. A professional camera would provide exact delays, the PS Eye however is not very accurate.

The angle detection algorithm has been optimized to perform even better than in the preliminary comparison shown in Figure 8. By introducing a rate limit, over 92% of samples leaves the sensor after exactly 8ms. This leads to the distribution shown in Figure 10.

When higher frame rates are applicable, bandwidth requirements must be considered: Even though only a floating point number is being transmitted, the sheer frequency of datagrams could congest some networks.

3.4.2. Serial Connection to the Actuator

To build an initial prototype, a serial connection using USB was implemented instead of an actual network. This is closer to the networking approach than to hardwired sensors, as a serial connection operates at a certain baud-rate and might use Cyclic Redundancy Checks (CRCs) as some types of packet switched networks do.

As shown in Figure 11, a serial connection does not add too much latency to the setup, but still is an overhead in comparison to a directly attached sensor. Therefore,

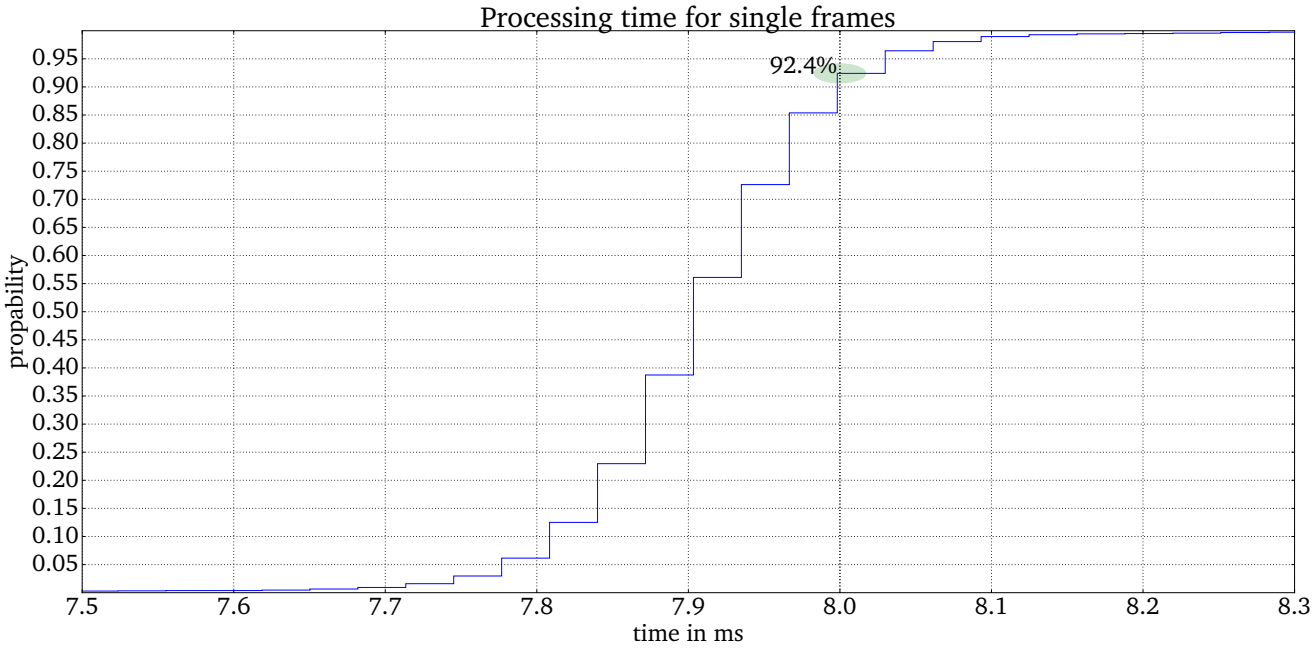


Figure 10: Cumulative Distribution Function (CDF) of the intervals between frames. Some frames are processed much faster than the nominal frame rate of 125Hz (8ms intervals between frames) suggests.

this represents a fair approximation of a very fast network and thus is a good starting point for degrading the network's quality for testing purposes.

Figure 11 also shows that the Baud rate can improve the performance and should be determined experimentally for the specific setup. A very high Baud rate leads to errors and thus retransmissions which explains the worse results with the highest rate shown in the plot. The Baud rate's impact on the latency is however small, as it mainly affects the possible bandwidth.

3.5. Controller Implementation

With the above setup, all aspects of the NCS are known and properly defined. For evaluation purposes, two controllers were implemented:

- A PID controller
- A LQR-based controller

While the PID controller is generally better known and easier to implement, it proved hard to stabilize by tuning its parameters manually. Furthermore, LQR-based controllers make it easier to control multiple variables with their cost matrices, which enables optimization of the carriage's position. In addition, LQR-based controllers can be designed with regard to eventual packet losses [BA12].

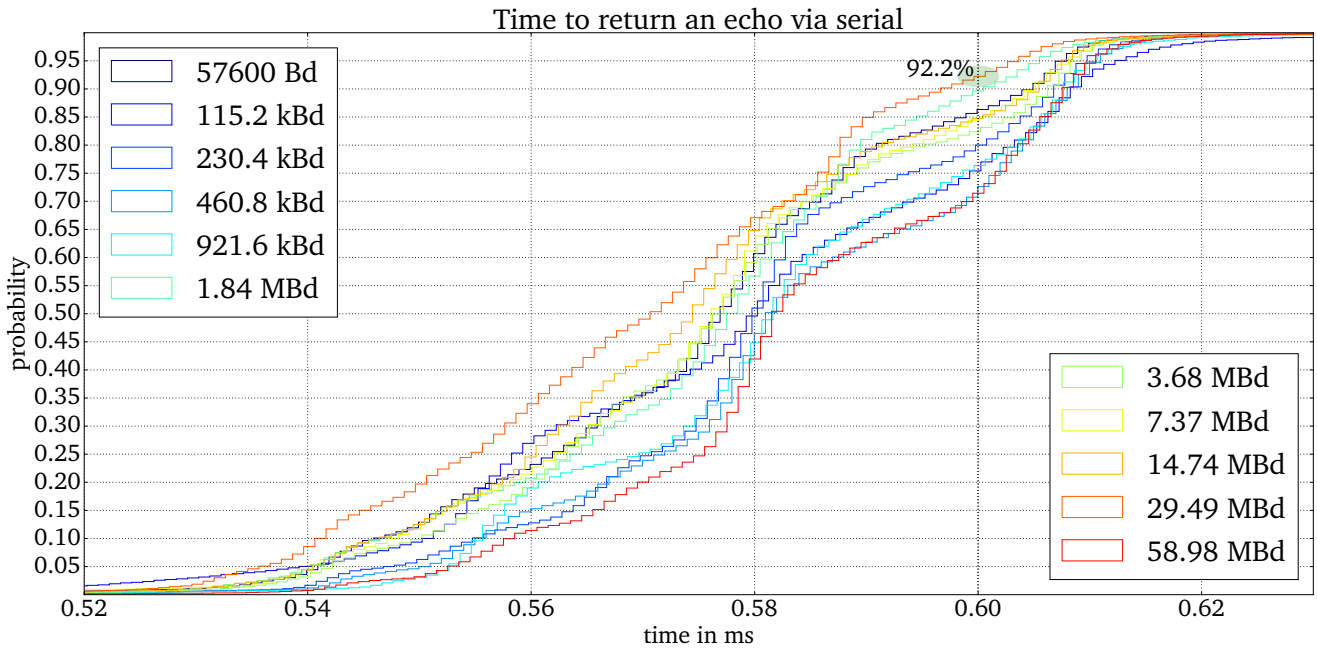


Figure 11: CDF of the RTTs of serial connections with varying baud rate. Measurements are made using an echo from the Arduino which is checked for transmission errors still present after the builtin CRC.

3.5.1. Theoretical Concepts of this Controller

As already explained in Section 2.3.2, the LQR problem is a optimization problem which minimizes a cost function (see Equation 2.3). This function contains two matrices Q and R, which define the cost of deviations from the zero vector and the cost of input respectively.

Using these matrices and the state space representation of the pendulum from Section 3.1.2, a discrete-time, infinite horizon gain matrix K is calculated using the algorithm specified by Blind et. al. [BA12].

With K defined, the controllers exact implementation is apparent from the following schematic:

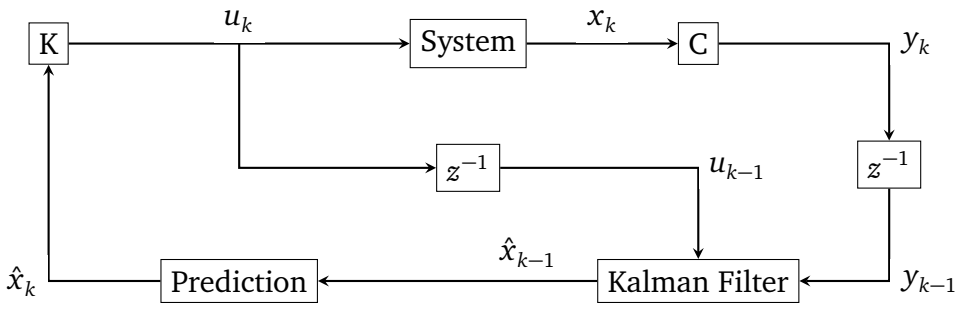


Figure 12: Schematic of the implemented controller.

To mitigate the delay present in the network, the sensors input and the last input into the system is filtered using a Kalman filter adapted for the possibility of packet losses

[BA12]. Generally, the use of a Kalman filter is a popular method to mitigate known delays which is used in different works [SFY09; Sin+04]. The Kalman filter outputs a estimation of the system's state during the last step by combining the information available from the previous two steps. Using this state estimation and the state space representation known from Section 3.1.2, it is possible to make an accurate prediction of the current system state: As the state space representation operates in a time-discrete manner, the state after another sampling period is easily determined as all variables are known.

This state is used to calculate the the acceleration as described in Section 3.5.3.

3.5.2. Simulation

Using the controller specified above, the system was simulated using GNU Octave [Pro15]. This simulation models the non-uniform delays as packet loss, because packets arriving after a deadline have to be considered lost. It does not matter when exactly packets arrive, as long it is before this deadline. Accordingly, this simulation has a concept of packet loss, but now of arrival time in general.

The full data of a simulation is visible in Figure 13. When the controller is performing bad, i.e. its weighted deviations are high and the correction performed is large, the pendulum's angle and the applied acceleration are at their maximum. Shortly after, the carriage position is at its maximum too. In the run plotted for this thesis no packet loss was assumed, as this represents the state when the network solution is working ideally. This meets the demonstrators goals, as its performance is expected to decline when exposed to non-uniform delays, i.e. packet loss.

The simulation results are conclusive and show that the designed controller can work with the demonstrator if an adequate network solution is inserted between sensor and actuator.

3.5.3. Implementation

The final implementation runs on the Arduino and is using a Interrupt Service Routine (ISR) triggered by a timer to accept and decode packets from the sensor. The extracted value is used to perform the prediction steps mentioned above.

The interval in which this timer triggers higher than both, the configured rate of new packets and the actuation interval of the stepper motor. This means that not every call to the ISR is used to receive sensor data or to maintain the carriages speed and acceleration for the whole sampling period. However, this cycle period presents a upper bound for both.

After the controller has a solid estimation of the current state, the acceleration needed to counteract any inclination or to return the carriage to the middle is calculated using the formula given by Equation 2.4. This acceleration is then converted to a interval using the formula defined in Equation 3.13 which in turn is converted to the number of cycle periods until the next actuation of the stepper motor or the next packet arrival.

Using this setup, the controller implementation performs according to the simulation given in Section 3.5.2.

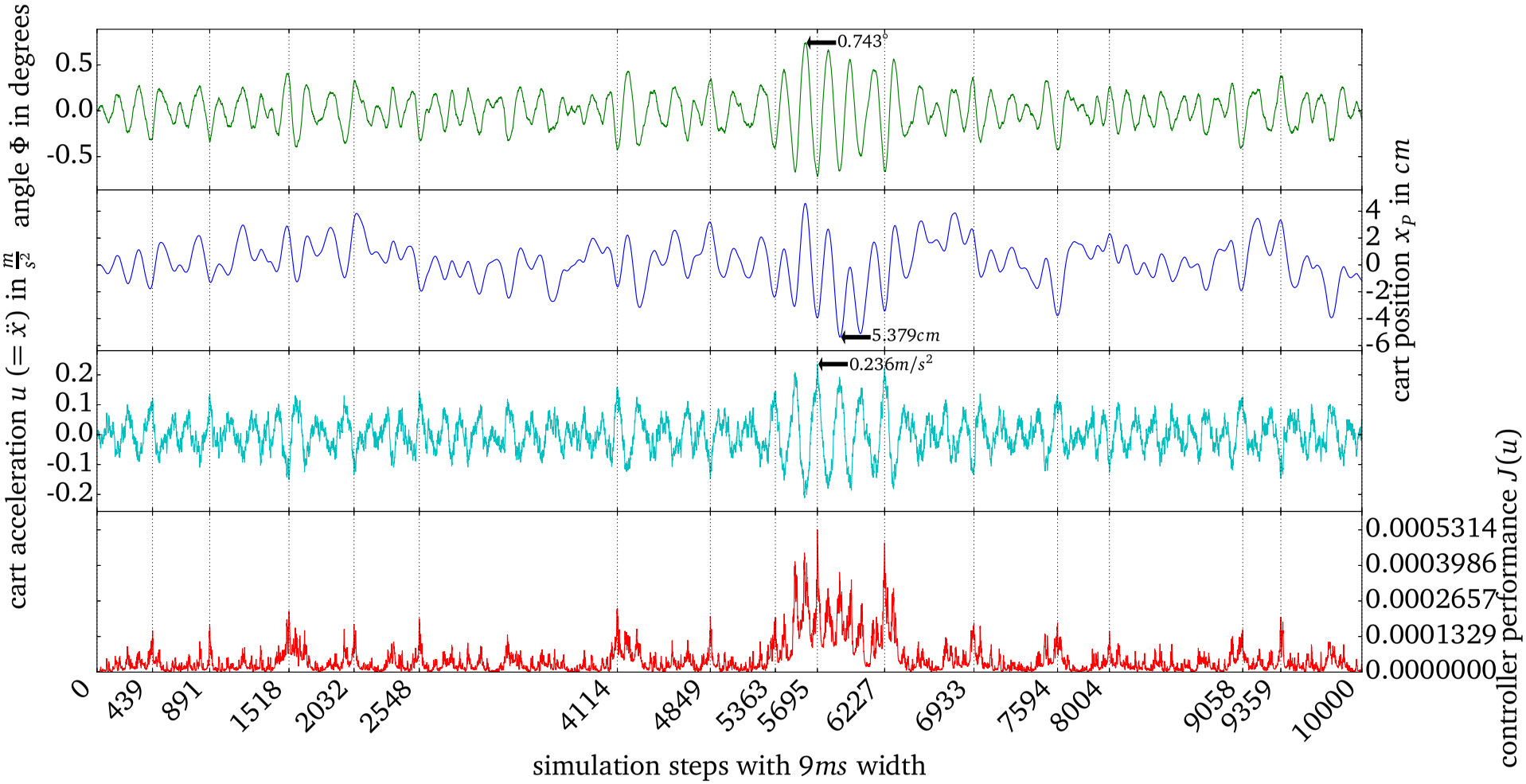


Figure 13: All relevant simulation data. The first three plots show physical properties of the pendulum at run time. The last plot visualizes the absolute cost which was optimized using a LQR (see Equation 2.3 from Section 2.3.2).

4. Closing the Feedback Loop over a Network

The testing of NCSs and making the results comparable is the main goal of this thesis. As such, the demonstrator is required to support as many intermediate networks as possible. Since the most widely used type of network is Ethernet, support for Ethernet and the protocols building upon it is imperative.

4.1. Networking with Arduino Due

The Arduino Due is easy to use but lacking in the area of compatibility with network interfaces. Its controller, the Atmel SAM3X8E [Atm15], features interfaces for SPI, CAN, and even an Ethernet Media Access Controller (EMAC), but lacks accessible pins for the latter.

4.1.1. Serial via USB

The default serial connection was already explained and measured in Section 3.4.2. To recap, the Round Trip Time (RTT) is 0.61*ms* or less in 99% of transmissions. This means likely one-way transmissions of less than 0.31*ms*, making it a sane default.

4.1.2. SPI and CAN

Both SPI and CAN are readily available for use on the Arduino Due. However, the sensor software of this demonstrator, i.e. the part running on the computer, does not implement these methods of communication, as there were no interfaces available for testing when writing this thesis.

Code to control these interfaces would be straightforward to write and easy to integrate with the existing sensor software.

4.1.3. Ethernet

The most universal interface available for the Arduino Due is Ethernet. Because Ethernet is well supported, the signal path can be arbitrarily extended using hardware commonly available or by using virtualization technology. To add such an interface to an Arduino Due, a so called Ethernet Shield is used. This is plugged onto the Arduino's SPI bus, allowing it to connect to a WIZNet W5100 Ethernet Controller [WC15a].

This setup however is very slow. This is mostly the fault of the aged W5100 chip which clocks its SPI interface at a maximum of 14.29MHz and does not allow the transmission of multiple bytes in a single transaction. The W5200 Ethernet Controller [WC15b] fixes these issues and should be able to provide much better speeds.

The delay subsequently is twice as high when comparing the RTT (best and worst case) of an Ethernet connection with that of a plain serial connection without a full IP stack (see Figures 11 and 14). While the current setup with the W5100 is generally slower than a serial connection, the newer W5200 chip should exceed the USB connections performance when a single link is used.

The cleanest and fastest variant would be to use the Ethernet MAC present on the boards processor, the Atmel SAM3X8E. This would require a physical layer component

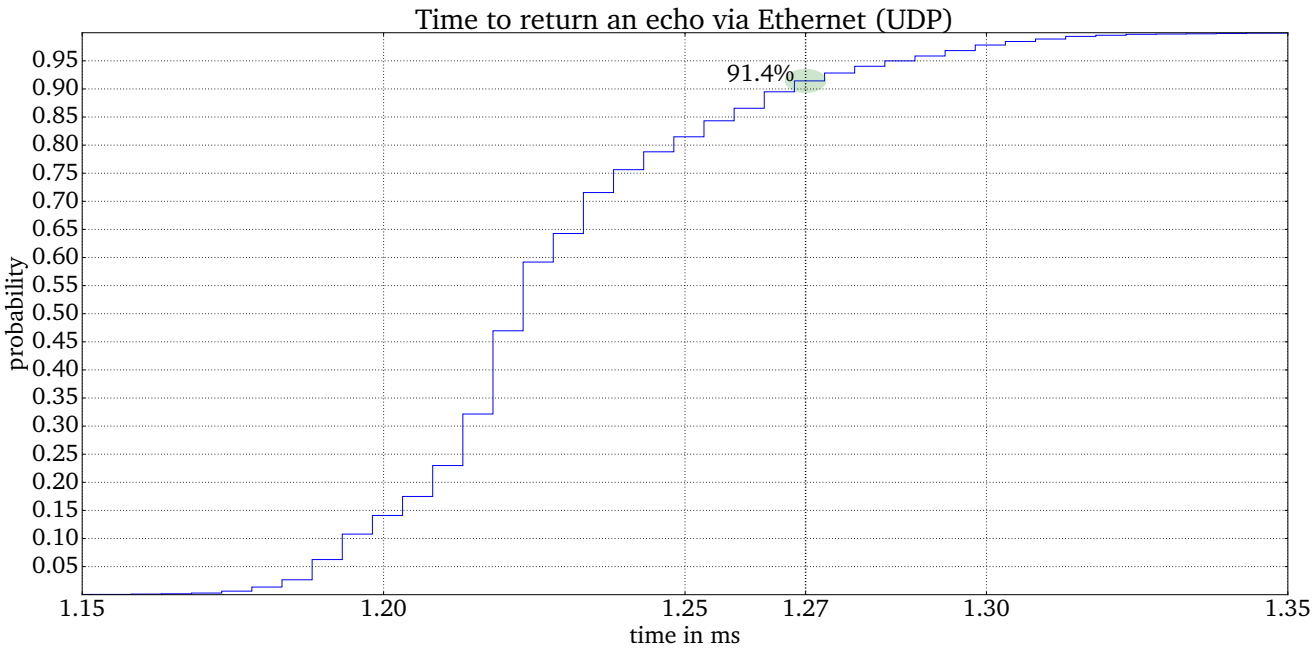


Figure 14: CDF of the RTTs in a UDP connection.

(PHY) for Ethernet. However, the corresponding pins are not broken out on the Arduino Due board which is why this option was not further explored. Many non-Arduino boards using the SAM3X8E expose the pins needed to operate the EMAC and have libraries to use Ethernet PHYs.

4.2. Mininet for LAN simulation

With an Ethernet interface in place, Mininet [Lan15] can be used to simulate an arbitrary, SDN controlled network. This is possible because one end of the transmission channel is a computer. A simulation with the physical link being the last link in the simulated network is easy and straightforward. Any other scenarios with the physical link in different locations are possible by using additional, intermediate hosts running Mininet. These however are unnecessary in most setups, as a NCS requires the whole network to be assessed, regardless of a single link which might have different properties.

As shown in Appendix D, the case in which the computer driving the camera is used for hosting the Mininet simulation is easy to realize and already provides a testing environment featuring arbitrary topologies and a SDN controller. In addition, the Mininet environment makes simulating degraded link quality and dynamic link outages easy.

Using the POX [Pox] controller in a simple OSI Layer 2 (L2) learning switch mode, the network worked flawlessly and introduced realistic latency and bandwidth properties when increasing the number of intermediate hosts.

4.3. Evaluation

With this setup in place, varying simulated networks behave as expected and can be adapted to test specific scenarios. This enables thorough testing of different NCS solutions based on using SDN technologies [Car+14] with arbitrary OpenFlow controllers.

However, this demonstrators only physical link does not offer a very good performance. As already touched on above, using the SAM3X8E's EMAC would be a more efficient way to implement Ethernet connectivity. An unofficial Arduino Due clone, like the TAIJIUINO Due Pro [Ele15], which has the EMAC pins broken out could be used to remedy these issues.

A good alternative would be a STM32 family controller (e.g. STM32F407 [STM15]) based development board like the Olimex STM32-E407 [Oli15]. It offers better overall performance and an on-board Ethernet interface at a lower price. The move toward a STM32 family board would however increase the complexity of the tool chain and the code required to operate the actuator hardware.

5. Summary

NCSs are always sensitive to changes in the networks quality of service. Many solutions for making NCSs viable exist, although they may use very different approaches.

The first possibility is to design a control system around the network. This is promising in many applications and works fine, as long as the network quality remains constant within certain bounds. The other possibility is to take any control system and make the network more predictable and capable to accommodate such a sensitive system.

5.1. Possibilities and Limitations of this Demonstrator

Any of these approaches often lack results based on actual, representative hardware tests which are easy to compare. An obvious difference between the two possibilities described above is the fact, that an easily applicable demonstrator is only viable for the second possibility, by making the intermediate network interchangeable. Specialized controllers always require rebuilding the setup for its specific requirements.

Therefore, this demonstrator provides the ability to test intermediate networks for NCSs and to compare the results.

5.1.1. Testing NCSs in varying Network Environments

This demonstrator offers interfaces for USB, SPI, CAN and Ethernet. While the latter use cases are obvious, USB offers only a serial connection at first. However, the demonstrator is using a Arduino Due which supports USB devices in host mode: This means that any interface for which a USB media converter is available can be supported. This is the case for several field bus systems, but also for different wireless solutions. On an even lower level, the Arduino also serves to interface with electronics directly, opening up a range of visual data transmission methods like infrared LEDs.

All these have to be implemented first which can be very time consuming for more exotic setups. However, many methods have already been explored by the Arduino Community and are readily available under open source licenses.

5.1.2. Testing NCSs in an OpenFlow-capable LAN

Testing different algorithms for SDN based networks proved straightforward and simple. Mininet makes changing setups easy and enables debugging at arbitrary levels of the networking stack. Any SDN controller supporting OpenFlow 1.0 or 1.3 works with Mininet and can be used to develop network applications providing an abstraction from low level networking, thus enabling the support of real time requirements needed by NCS.

5.1.3. Testing specialized NCS Controllers

As explained above, testing specialized NCS controllers is not a viable option with an unified demonstrator such as this. This is because much of the setup detailed above has to be reimplemented using the NCS controller. In such cases it is easier to built a dedicated prototype instead of targeting the platform chosen for another setup.

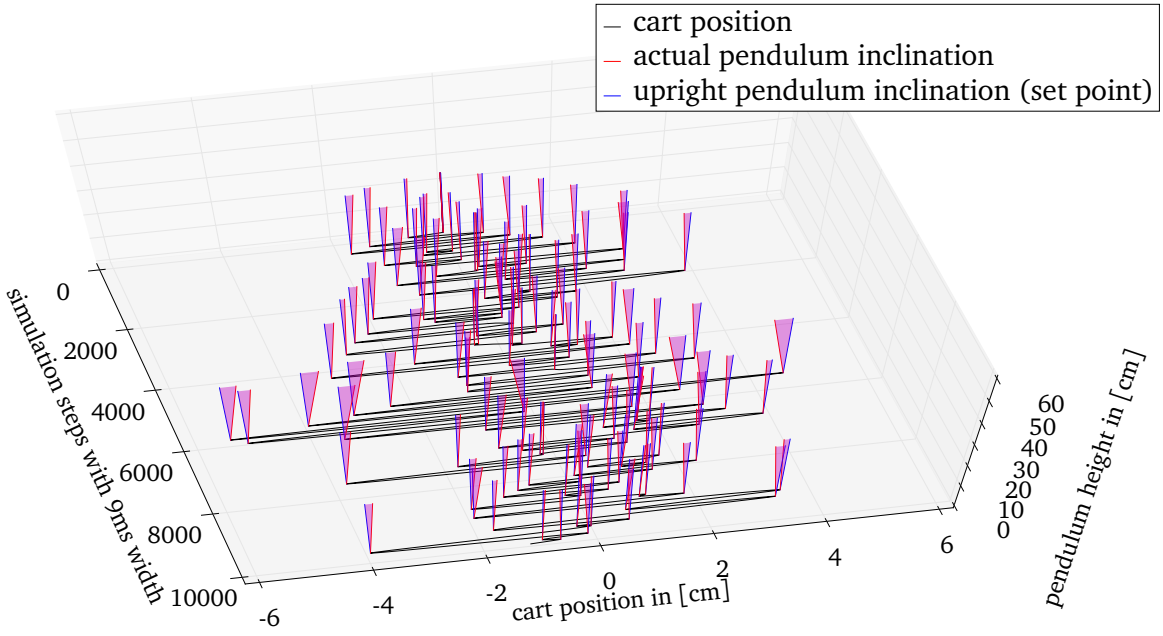


Figure 15: Visualization of the simulation. This is rendered using the same data as Figure 13. The shown angles are scaled to be more visible.

However, the inverted pendulum is the single most studied control system and also a prime example for a closed-loop system. When implementing a showcase implementation, the inverted pendulum should be considered with a strong preference. In addition, using a camera as the sensor makes for a convincing setup which is not only constructed from the need to have actuators and sensors on different network nodes.

This demonstrator's sensor software is reusable, well optimized and uses very popular, consumer-grade hardware. If a camera is chosen as sensor, this can save the effort of writing dedicated sensor software.

5.2. Future Work

Even if the current state of this demonstrator provides a basic environment for testing and comparing NCS optimizations and algorithms in the network, there is much room for improvements.

The current controller can and should be stabilized further. It is apparent from the simulation in Section 3.5.2 that the controller is far from perfect. Even though the set point deviations are small, the cart is moving around too much. This also leads to a poor robustness against manual disturbances. For a visualization of this, please see Figure 15.

While the sensor software and the controller are disjoint for now, a major addition would be the introduction of a negotiation protocol. Right now, an upper bound for the delay on the feedback loop is configured statically in the controller. Having the sensor software and the controller negotiate this upper bound dynamically could prove to be beneficial in certain cases. Also, the introduction of a lower bound for transmission de-

lay can increase a controller's efficiency. These two measures could further improve or even replace the Kalman filter and the prediction step present in front of the controller implementation.

As noted in Section 4.3, the Ethernet interface used to connect the Arduino did not meet expectations. The next step would be to use a more sophisticated controller board than the Arduino Due which also meets industrial quality requirements and thus is a more realistic demonstration. Any STM32F407 family [STM15] development board with an Ethernet PHY presents a promising candidate which could serve as a drop in replacement for the Arduino Due and the Ethernet Shield.

5.3. Conclusion

The study of NCSs is as relevant as ever. The networking requirements are easily matched by today's hardware and the applications profiting from NCSs are steadily growing. Hence, research in this area continues to produce solutions to the problems NCSs pose.

These solutions are divided into those which optimize the controllers to work with inconsistent communication, and those which mitigate delays within the network to properly serve an unoptimized controller. The goal of this thesis was to present the setup of a demonstrator which makes testing of the solutions acting in the network possible and the results comparable.

The first section of this thesis described the need for a demonstrator for NCSs and suggests the use of the well-known and thoroughly documented inverted pendulum. After summarizing the background needed to built this control system in Section 2, the third section detailed the setup of an inverted pendulum with visual angle detection, a very practical and representative example for NCSs. This thesis concluded with a description of different networking interfaces which can be used with the resulting demonstrator.

All in all, a setup consisting of commodity hardware was proven to be sufficient to built a operational demonstrator for NCSs. The sensor software needed to use such a camera to detect the pendulums inclination was written and evaluated, proving it stable in its measurements and its performance. The actuator software was realized using a well known micro-stepping driver and can be reused for any future NCS demonstrator setup. The communication interface between sensor and actuator was kept as variable as possible while exploring the possibilities of a connection via virtualized Ethernet networks.

Finally, a discrete-time system model was derived and an appropriate, LQR based controller was suggested. This controller's simulation and implementation have been explained in detail to ensure the reproducibility of this setup and to enable the testing of arbitrary solutions to NCS problems.

Appendices

A. Solving the Inverted Pendulum's Lagrangian

To recap, these are the definitions from Section 3.1.1, with (x_P, y_P) and (x_M, y_M) being the Cartesian coordinates of the pivot P and the pendulum's center of mass M, respectively. :

$$\begin{aligned} x_P &= x & \dot{x}_P &= \dot{x} \\ y_P &= 0 & \dot{y}_P &= 0 \\ x_M &= x - \frac{l}{2} \cdot \sin(\Phi) & \dot{x}_M &= \dot{x} - d\dot{\Phi}\cos\Phi \\ y_M &= \frac{l}{2} \cdot \cos(\Phi) & \dot{y}_M &= -d\dot{\Phi}\sin\Phi \end{aligned}$$

They define the positions of the pendulum's base P and center of mass M (see Figure 4) and their derivatives, i.e. their speed.

These yield the kinetic energy T and the potential energy V present this system and thus, its Lagrangian ($L = T - V$):

$$v_M^2 = \dot{x}_M^2 + \dot{y}_M^2 = \dot{x}_M^2 - d\dot{x}_P\dot{\Phi}\cos\Phi + d^2\dot{\Phi}^2 \tag{A.1}$$

$$T = \frac{1}{2}mv_M^2 + \frac{1}{2}I_P\dot{\Phi}^2 \tag{A.2}$$

$$V = m \cdot g \cdot y_M \tag{A.3}$$

$$L = T - V \tag{A.4}$$

$$= \left(\frac{1}{2}mv_M^2 + \frac{1}{2}I_P\dot{\Phi}^2\right) - (m \cdot g \cdot y_M) \tag{A.5}$$

$$= \left(\frac{1}{2}m\left(\dot{x}_M^2 - d\dot{x}_P\dot{\Phi}\cos\Phi + d^2\dot{\Phi}^2\right) + \frac{1}{2}\left(\frac{1}{3}ml^2\right)\dot{\Phi}^2\right) + \left(mgd\dot{\Phi}\sin\Phi\right) \tag{A.6}$$

$$= m\left(\frac{1}{2}\dot{x}_M^2 - d\dot{x}_P\dot{\Phi}\cos\Phi + \frac{1}{2}d^2\dot{\Phi}^2\right) + m\left(\frac{1}{6}l^2\dot{\Phi}^2 + gd\dot{\Phi}\sin\Phi\right) \tag{A.7}$$

$$L = m\left(\frac{1}{2}\dot{x}_M^2 + \frac{1}{2}\left(d^2 + \frac{1}{3}l^2\right)\dot{\Phi}^2 - d\left(\dot{x}_P\dot{\Phi} + g\right)\cos\Phi\right) \tag{A.8}$$

This equation can be solved for $\ddot{\Phi}$ by using the Euler-Lagrange equation:

$$\frac{d}{dt}\frac{\partial L}{\partial\dot{\Phi}} = \frac{\partial L}{\partial\Phi} \tag{A.9}$$

$$\frac{\partial L}{\partial\dot{\Phi}} = \left(d^2 + \frac{1}{3}l^2\right)\dot{\Phi} - d\dot{x}_P\cos(\Phi) \tag{A.10}$$

$$\frac{\partial L}{\partial\Phi} = d\left(\dot{x}_P\dot{\Phi} + g\right)\sin(\Phi) \tag{A.11}$$

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\Phi}} - \frac{\partial L}{\partial \Phi} = 0 \tag{A.12}$$

$$\frac{d}{dt}\left(\left(d^2 + \frac{1}{3}l^2\right)\dot{\Phi} - d\dot{x}_P\cos(\Phi)\right) - d\left(\dot{x}_P\dot{\Phi} + g\right)\sin(\Phi) = 0 \tag{A.13}$$

$$\left(d^2 + \frac{1}{3}l^2\right)\ddot{\Phi} - d\ddot{x}_P\cos(\Phi) + \cancel{d\dot{x}_P\dot{\Phi}\sin(\Phi)} - d\left(\cancel{\dot{x}_P\dot{\Phi}} + g\right)\sin(\Phi) = 0 \tag{A.14}$$

$$\Rightarrow \quad \ddot{\Phi} = \underbrace{\frac{d}{d^2 + \frac{1}{3}l^2}}_{=F,\text{ a constant factor}}\left(\ddot{x}_p\cos(\Phi) + g\sin(\Phi)\right) \tag{A.15}$$

Which is the equation for $\ddot{\Phi}$ given in Section 3.1.1.

B. Discretization of a continuous-time State Space Representation

The discrete time state space representation of the system is given by the following equations:

$$x_{k+1} = A_d x_k + B_d u_k \tag{B.1}$$

$$y_k = C x_k \tag{B.2}$$

To obtain A_d and B_d from their continuous-time variants, a general solution to the first order differential $\dot{x} = A \cdot x + B \cdot u$ is used and discretized according to the method used by Blind et. al. [BA12]:

$$x(t) = e^{At} \cdot x(t_0) + \int_{t_0}^{t}\left(e^{A\tau} \cdot Bu(\tau)\right)d\tau \tag{B.3}$$

$$x_{k+1} = \underbrace{e^{AT_s}}_{A_d} x_k + \underbrace{\int_0^{T_s}\left(e^{A\tau} \cdot B\right)d\tau}_{B_d} \cdot u_k \tag{B.4}$$

C. OpenCV Angle Detection

The classes below present a simplified version of the algorithm used to detect the pendulums inclination. As it operates only on two markers, the camera has to be setup very precisely. The full code is available with the electronic release of this thesis. Refer to inline comments for further details.

The following class is simplification for the class below. It updates color ranges according to the track bars displayed by the GUI, offering the results to the main algorithm.

```
class Marker(object):
    """ A class for identifying objects by defining a range of colors with
     cv2 track bars.

    A Marker defines an upper and a lower bound for each value within the HSV
    color space and can be used for filtering a HSV image for a specific color,
    i.e. an object (the marker) with that color, if it is sufficiently unique
    in the overall picture.

    """
    PARAMETERS = [("hue", 179), ("sat", 255), ("val", 255)]

    def __init__(self, name, attach_window="Trackbars"):
        """ Constructor, initialize Trackbars. """
        self.name = name
        self.attach_window = attach_window

        def update(data):
            """ unused callback """
            pass

        for param_name, maximum in self.PARAMETERS:
            cv2.createTrackbar("l{0}_{1}".format(param_name, name), attach_window, 0, maximum, update)
            cv2.createTrackbar("h{0}_{1}".format(param_name, name), attach_window, maximum, maximum, update)

    def get_limits(self):
        lower_vector = np.array([cv2.getTrackbarPos("l{0}_{1}".format(param_name, self.name), self.attach_window) for param_name, _ in self.PARAMETERS], np.uint8)
        upper_vector = np.array([cv2.getTrackbarPos("h{0}_{1}".format(param_name, self.name), self.attach_window) for param_name, _ in self.PARAMETERS], np.uint8)
        return lower_vector, upper_vector
```

The following class contains the main mechanism of angle detection.

```
class AngleDetector(object):
    """ A class for calculating the angle of the line between two points to the
        horizon.

    It calculates the angle between two markers defined by Marker objects. The
    angle is returned in radians in the interval [-pi,pi], with 0 being a
    upright position. "upright" assumes that the first marker (a) is at the
    bottom.
    """

    def __init__(self, cap_device, resolution=(320, 240), fps=125, output=True):
        """ Constructor, initialize camera and GUI """
        self.capture = cv2.VideoCapture(cap_device)
        self.resolution = resolution
        self.fps = fps
        self.output = output

        self.capture.set(cv2.cv.CV_CAP_PROP_FRAME_WIDTH, resolution[0])
        self.capture.set(cv2.cv.CV_CAP_PROP_FRAME_HEIGHT, resolution[1])
        self.capture.set(cv2.cv.CV_CAP_PROP_FPS, fps)

        if self.output:
            cv2.namedWindow("Angle")
            cv2.namedWindow("Threshold A (Pivot)")
            cv2.namedWindow("Threshold B (Tip)")
            cv2.namedWindow("Trackbars")

        self.marker_names = ('a', 'b')
        self.markers = (Marker(self.marker_names[0], attach_window="Trackbars"), Marker(self.marker_names[1], attach_window="Trackbars"))

    def __del__(self):
        cv2.destroyAllWindows()
        self.capture.release()

    def find_marker(self, hsv_img, marker):
        """ This methods finds any markers defined by the program and returns the binary image as
        well as a set of coordinates """
        lower_vector, upper_vector = marker.get_limits()
```

```
39
40          threshold_img = cv2.inRange(hsv_img, lower_vector,
               ↪ upper_vector)
41          image = threshold_img.copy()
42
43          contours, _ = cv2.findContours(threshold_img, cv2.
               ↪ RETR_LIST, cv2.CHAIN_APPROX_NONE)
44          best_contour = None
45          max_area = 0
46          for contour in contours:
47              area = cv2.contourArea(contour)
48              if area > max_area:
49                  max_area = area
50                  best_contour = contour
51          x = 0
52          y = 0
53          if best_contour is not None:
54              moments = cv2.moments(best_contour)
55              area = moments['m00']
56              x = int(moments['m10'] / area)
57              y = int(moments['m01'] / area)
58          return image, (x, y)
59
60      def get_current_angle(self):
61          """ Main angle detection routine. this method is
               ↪ called by the main program and returns the
62          angle on basis of the markers found in the picture.
               ↪  """
63          # capture a image and convert to hsv
64          _, img = self.capture.read()
65          hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
66
67          marker_img = dict()
68          marker_coords = dict()
69
70          for marker in self.markers:
71              marker_img[marker.name], marker_coords[marker.
                   ↪ name] = self.find_marker(hsv_img, marker)
72              cv2.circle(img, marker_coords[marker.name], 2,
                   ↪ (0, 255, 0), 20)
73
74          x1, y1 = marker_coords[self.marker_names[0]]
75          x2, y2 = marker_coords[self.marker_names[1]]
76          if self.output:
77              cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0),
                   ↪ 3, cv2.CV_AA)
78              cv2.line(img, (x1, y1), (img.shape[1], y1),
                   ↪ (255, 0, 0), 3, cv2.CV_AA)
79
```

```
80          # calculate the angle
81          if x1 == x2: # this is the upright case
82              angle = 0.0
83          elif y1 == y2: # this is the horizontal case
84              if x2 < x1:
85                  angle = -np.pi / 2
86              else:
87                  angle = np.pi / 2
88          else:
89              angle = np.arctan(float(y2 - y1)/float(x2 - x1)
                    ↪ ) + np.pi/2
90              # rotate the angle, so a upright position
                    ↪ equals 0
91              if x2 < x1:
92                  angle -= np.pi
93
94          if self.output:
95              # put the angle on the output picture
96              cv2.putText(img, str(angle), (20, img.shape[0]
                    ↪ - 20), cv2.FONT_HERSHEY_SIMPLEX, 1.0,
                    ↪ (255, 255, 255), 2)
97              # display frames to users
98              cv2.imshow("Angle", img)
99              cv2.imshow("Threshold A (Pivot)", marker_img['a
                    ↪ '])
100             cv2.imshow("Threshold B (Tip)", marker_img['b'
                    ↪ ])
101         return angle
```

D. Mininet Configuration

The following lines of Python [Ros15] code attach a hardware interface to the Mininet [Lan15] simulation. Refer to inline comments for more details.

```
#!/usr/bin/python

# import required libraries
from mininet.cli import CLI
from mininet.net import Mininet
from mininet.link import TCIntf
from mininet.topolib import TreeTopo

if __name__ == '__main__':
    # name of the hardware interface
    iface_name = 'eth0'

    # use a binary tree of depth three as topology
    net = Mininet(topo=TreeTopo(depth=3, fanout=2))

    # take the root switch and attach the hardware to it
    switch = net.switches[0]
    iface = TCIntf(iface_name, node=switch)

    # optionally, specify properties of this interface
    iface.config(loss=50)  # packet loss in %
    iface.config(delay=15) # additional delay in ms
    iface.config(bw=1)     # bandwidth in Mb/s

    # start, show cli, and clean up
    net.start()
    CLI(net)
    net.stop()
```

With this simulation running, an arbitrary SDN controller like POX [Pox] can be started to fill the switches flow tables. Of course, the Mininet configuration has to be adjusted to find the controller when using another port than 6633 or another host than localhost.

With this in place, the demonstrator can be connected to the interface specified in the above code. The sensor software needs to be started from one of the Mininet hosts by issuing *h1 xterm* followed by the setup required for the sensor.

References

[All09] Allegro. *Allegro A4988 Microstepping Driver Datasheet*. 2009. URL: http://www.allegromicro.com/~/Media/Files/Datasheets/A4988-Datasheet.ashx.

[AS03] Babak Azimi-Sadjadi. "Stability of networked control systems in the presence of packet losses". In: *Proceedings of 2003 IEEE Conference on Decision and Control*. Vol. 1. 2003, pp. 676–681. URL: http://scl.hanyang.ac.kr/scl/database/papers/CDC/2003/pdffiles/papers/tum06.2.pdf.

[Ass15] Open DeviceNet Vendor Association. *DeviceNet*. 2015. URL: https://www.odva.org/Home/ODVATECHNOLOGIES/DeviceNet/DeviceNetTechnologyOverview.aspx.

[Atm15] Atmel. *Atmel SAM3X8E Microcontroller Datasheet*. 2015. URL: http://www.atmel.com/Images/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A_Datasheet.pdf.

[BA11a] Rainer Blind and Frank Allgöwer. "Analysis of Networked Event-Based Control with a Shared Communication Medium: Part I-Pure ALOHA". In: *Jeb* 1 (2011), p. 2. URL: http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac11-proceedings/data/html/papers/1100.pdf.

[BA11b] Rainer Blind and Frank Allgöwer. "On the optimal sending rate for networked control systems with a shared communication medium". In: *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*. IEEE, 2011, pp. 4704–4709. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6160886.

[BA12] Rainer Blind and F. Allgower. "Is it worth to retransmit lost packets in Networked Control Systems?" In: *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*. IEEE, 2012, pp. 1368–1373. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6426881.

[BA14] Rainer Blind and Frank Allgöwer. "On the stabilizability of continuous-time systems over a packet based communication system with loss and delay". In: *World Congress of the International Federation of Automatic Control (IFAC)*. 2014. URL: http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2014/media/files/1213.pdf.

[BPZ00] Michael S. Branicky, Stephen M. Phillips, and Wei Zhang. "Stability of networked control systems: Explicit analysis of delay". In: *American Control Conference, 2000. Proceedings of the 2000*. Vol. 4. IEEE, 2000, pp. 2352–2357. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=878601.

[Car+14] Ben William Carabelli et al. *A Network Abstraction for Control Systems*. Technical Report 2014/01, University of Stuttgart, Institute for Parallel and Distributed Systems, 2014. URL: ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/TR-2014-01/TR-2014-01.pdf.

[e.V15] PROFIBUS Nutzerorganisation e.V. *ProfiBus*. 2015. URL: http://www.profibus.com/technology/profibus/.

[Ele15] ElecHouse. *TAIJIUINO Due Pro R2 Information Page*. 2015. URL: http://www.elechouse.com/elechouse/index.php?main_page=product_info&cPath=72_73&products_id=2224.

[Gar15] Willow Garage. *OpenCV*. 2015. URL: http://opencv.org/.

[GC10] Rachana Ashok Gupta and Mo-Yuen Chow. "Networked control system: overview and research trends". In: *Industrial Electronics, IEEE Transactions on* 57.7 (2010), pp. 2527–2535. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5313866.

[Hee+09] WPMH Heemels et al. "networked control systems with communication constraints: Tradeoffs between transmission intervals and delays". In: *Control Conference (ECC), 2009 European*. IEEE, 2009, pp. 4296–4301. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=7075075.

[HNX07] Joao P. Hespanha, Payam Naghshtabrizi, and Yonggang Xu. "A survey of recent results in networked control systems". In: *PROCEEDINGS-IEEE* 95.1 (2007), p. 138. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.216.9432&rep=rep1&type=pdf.

[Hou62] P. V. C. Hough. *Method and Means for Recognizing Complex Patterns*. US 3069654. Originating Research Org. not identified, Dec. 18, 1962. URL: http://www.osti.gov/scitech/biblio/4746348.

[How14] Joe Howse. *Linux-Kernel Patch to gspca: Add high-speed modes for PS3 Eye camera*. 2014. URL: http://lkml.iu.edu/hypermail/linux/kernel/1412.3/01236.html.

[Hu62] Ming-Kuei Hu. "Visual pattern recognition by moment invariants". In: *IRE Transactions on Information Theory* 8.2 (Feb. 1962), pp. 179–187. ISSN: 0096-1000. DOI: 10.1109/TIT.1962.1057692.

[Kir09] Peter Kirn. *Trick Out Your PS3 Eye Webcam, Best Cam for Vision, Augmented Reality*. 2009. URL: http://createdigitalmotion.com/2009/08/trick-out-your-ps3-eye-webcam-best-cam-for-vision-augmented-reality/.

[Lan15] Bob Lantz. *Mininet*. 2015. URL: http://mininet.org/.

[Lea] *Leap Motion Controller*. 2015. URL: https://www.leapmotion.com/.

[LMT02] Feng-Li Lian, James Moyne, and Dawn Tilbury. “Network design consideration for distributed control systems”. In: *Control Systems Technology, IEEE Transactions on* 10.2 (2002), pp. 297–307. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=987076.

[MDF00] T. O. Mahony, C. J. Downing, and K. Fatla. “Genetic algorithm for PID parameter optimization: minimizing error criteria”. In: *Process control and instrumentation* (2000), pp. 26–28.

[NAR08] A. N. K. Nasir, M. A. Ahmad, and M. F. Rahmat. “Performance Comparison Between Lqr and Pid Controllers for an Inverted Pendulum System”. In: *AIP Conference Proceedings* 1052.1 (Oct. 7, 2008), pp. 124–128. ISSN: 0094243X. DOI: 10.1063/1.3008655.

[Nis00] Norman S. Nise. *Control Systems Engineering*. 3rd. New York, NY, USA: John Wiley & Sons, Inc., 2000. ISBN: 0-471-36601-3.

[No98] Johan Nilsson and others. “Real-time control systems with delays”. PhD thesis. Lund institute of Technology Lund, Sweden, 1998. URL: http://www.cds.caltech.edu/~murray/wiki/images/8/8b/Nilsson_thesis_98.pdf.

[Oli15] Olimex. *STM32-E407 - Open Source Hardware Board*. 2015. URL: https://www.olimex.com/Products/ARM/ST/STM32-E407/.

[PFA05] Kaustubh Pathak, Jaume Franch, and Sunil K. Agrawal. “Velocity and position control of a wheeled inverted pendulum by partial feedback linearization”. In: *Robotics, IEEE Transactions on* 21.3 (2005), pp. 505–513. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1435497.

[Pic] *Raspberry Pi Camera*. 2015. URL: https://www.raspberrypi.org/products/camera-module/.

[Pox] *POX Controller*. 2015. URL: http://www.noxrepo.org/.

[Pro15] The GNU Project. *GNU Octave*. 2015. URL: https://www.gnu.org/software/octave/.

[Ric03] Jean-Pierre Richard. “Time-delay systems: an overview of some recent advances and open problems”. In: *automatica* 39.10 (2003), pp. 1667–1694. URL: http://www.sciencedirect.com/science/article/pii/S0005109803001675.

[RK08] Craig L. Robinson and P. R. Kumar. “Optimizing controller location in networked control systems with packet drops”. In: *Selected Areas in Communications, IEEE Journal on* 26.4 (2008), pp. 661–671. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4497791.

[Ros15] Guido van Rossum. *Python*. 2015. URL: https://www.python.org/.

[SFY09] Y. Shi, H. Fang, and M. Yan. “Kalman filter-based adaptive control for networked systems with unknown parameters and randomly missing outputs”. In: *International Journal of Robust and Nonlinear Control* 19.18 (2009), pp. 1976–1992. URL: http://onlinelibrary.wiley.com/doi/10.1002/rnc.1390/full.

[Sin+04] Bruno Sinopoli et al. “Kalman filtering with intermittent observations”. In: *Automatic Control, IEEE Transactions on* 49.9 (2004), pp. 1453–1464. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1333199.

[SQ03] Hu Shousong and Zhu Qixin. “Stochastic optimal control and analysis of stability of networked control systems with long delay”. In: *Automatica* 39.11 (2003), pp. 1877–1884. URL: http://www.sciencedirect.com/science/article/pii/S0005109803001961.

[STM15] STMicroelectronics. *STM32F407/417 - STMicroelectronics*. 2015. URL: http://www.st.com/web/en/catalog/mmc/FM141/SC1169/SS1577/LN11.

[TC03] Yodyium Tipsuwan and Mo-Yuen Chow. “Control methodologies in networked control systems”. In: *Control engineering practice* 11.10 (2003), pp. 1099–1111. URL: http://www.sciencedirect.com/science/article/pii/S0967066103000364.

[Wan+10] Hongliang Wang et al. “Design and Simulation of LQR Controller with the Linear Inverted Pendulum”. In: *2010 International Conference on Electrical and Control Engineering (ICECE)*. 2010 International Conference on Electrical and Control Engineering (ICECE). June 2010, pp. 699–702. DOI: 10.1109/iCECE.2010.178.

[WC15a] Ltd. WIZnet Co. *W5100 Datasheet*. 2015. URL: http://www.wiznet.co.kr/product-item/w5100/.

[WC15b] Ltd. WIZnet Co. *W5200 Datasheet*. 2015. URL: http://www.wiznet.co.kr/product-item/w5200/.

[WZC00] Wei Wang, Jingtao Zhang, and Tianyou Chai. “A survey of advanced PID parameter tuning methods”. In: *Acta Automatica Sinica* 26.3 (2000), pp. 347–355.

[Yam89] Takeshi Yamakawa. “Stabilization of an inverted pendulum by a high-speed fuzzy logic controller hardware system”. In: *Fuzzy sets and Systems* 32.2 (1989), pp. 161–180. URL: http://www.sciencedirect.com/science/article/pii/0165011489902522.

[Yan06] Tai C. Yang. “Networked control system: a brief survey”. In: *IEE Proceedings-Control Theory and Applications* 153.4 (2006), pp. 403–412. URL: http://digital-library.theiet.org/content/journals/10.1049/ip-cta_20050178.

[Yur08] V.D. Yurkevich. “PI and PID controller design for nonlinear systems in the presence of a time delay via singular perturbation technique”. In: *9th International Conference on Actual Problems of Electronic Instrument Engineering, 2008. APEIE 2008*. 9th International Conference on Actual Problems of Electronic Instrument Engineering, 2008. APEIE 2008. Vol. 01. Sept. 2008, pp. 168–174. DOI: 10.1109/APEIE.2008.4897078.

[YZH05] Zhi Yang, Hai-feng Zhu, and Yi-hua Huang. “Recent Studies of PID Design and Parameter Tuning Method”. In: *Control and Instruments In Chemical Industry* 32.5 (2005), p. 1. URL: http://en.cnki.com.cn/Article_en/CJFDTotal-HGZD200505000.htm.

[ZBP01] Wei Zhang, Michael S. Branicky, and Stephen M. Phillips. “Stability of networked control systems”. In: *Control Systems, IEEE* 21.1 (2001), pp. 84–99. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=898794.

[Zha+05] Liqian Zhang et al. “A new method for stabilization of networked control systems with random delays”. In: *Automatic Control, IEEE Transactions on* 50.8 (2005), pp. 1177–1181. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1492560.

[LLC15] Arduino LLC. *Arduino - ArduinoBoardDue*. 2015. URL: http://www.arduino.cc/en/Main/ArduinoBoardDue.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature