BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF CONTROL AND INSTRUMENTATION
ÚSTAV AUTOMATIZACE A MĚŘÍCÍ TECHNIKY

INVERTED PENDULUM REMOTE REAL-TIME CONTROLLER IMPLEMENTED IN MATLAB-SIMULINK AND LABVIEW ENVIRONMENTS

REGULÁTOR INVERZNÍHO KYVADLA IMPLEMENTOVANÝ V REÁLNÉM ČASE V PROSTŘEDÍ MATLAB-SIMULINK A LABVIEW

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — Jan Mohyla
AUTOR PRÁCE

SUPERVISOR — prof. Ing. Pavel Václavek, Ph.D.
VEDOUCÍ PRÁCE

BRNO 2016

BRNO UNIVERSITY OF TECHNOLOGY

Faculty of Electrical Engineering and Communication

Department of Control and Instrumentation

Bachelor thesis

Bachelor's study field
Automation and Measurement

Student: Jan Mohyla ***ID:*** 164344
Year of study: 3 ***Academic year:*** 2015/16

TITLE OF THESIS:

Inverted Pendulum Remote Real-time Controller Implemented in Matlab-Simulink and LabView Environments

INSTRUCTION:

The goal of the thesis is to study integration of Matlab implemented controller and real inverted pendulum process attached to National Instruments Compact RIO system. The developed system will be used to demonstrate variable delay effect on control loop. Key problems to be solved are:
1. Developing a controller for the inverted pendulum, running the Simulink controller and communication in real time;
2. Developing a real time communication protocol between Simulink controller and LabView program running on Compact RIO and developing a LabView program interfacing with the physical system;
3. As a proof of proposed concept a demonstration of proposed systems will be developed.

REFERENCE:

J.J.C. van Schendel, Networked Control Systems -Simulation & Analysis, 2008

Assigment deadline: 8. 2. 2016 ***Submission deadline:*** 23. 5. 2016

Head of thesis: prof. Ing. Pavel Václavek, Ph.D.
Consultant:

doc. Ing. Václav Jirsík, CSc.
Subject Council chairman

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ Fakulta elektrotechniky a komunikačních technologií Ústav automatizace a měřicí techniky

WARNING:

ABSTRACT

This Bachelor's thesis focuses on the remote control demonstrated on the rotational inverted pendulum. The objective of this project is to design the control of an inverted pendulum in Simulink environment and to develop a real-time communication protocol between the controller created in Simulink and the LabView software running on CompactRIO which will be interconnected with a physical system. The role of the controller is to maintain the pendulum upright not to swing the pendulum upright - the pendulum is swung up manually.

KEYWORDS

Rotary inverted pendulum, remote control, data exchange between Simulink and LabView, network control system, system with delay, LQR regulator

ABSTRAKT

V této bakalářské práci bude diskutováno vzdálené řízení demonstrováno na rotačním inverzním kyvadle. Hlavním cílem je navrhnout řízení inverzního kyvadla v Simulinku a zajistit komunikaci v reálném čase mezi regulátorem vytvořeném v Simulinku a LabView programem, běžícím na CompactRIO, který bude propojený s fyzickým systémem. Úkol regulátoru je pouze udržet inverzní kyvadlo ve vzpřímené poloze, nikoli vyšvihnutí do vzpřímené polohy (bude provedeno ručně).

KLÍČOVÁ SLOVA

Rotační inverzní kyvadlo, vzdálené řízení, výměna dat mezi Simulinkem a LabView, systém se spožděním, LQR regulátor

MOHYLA, Jan *Inverted Pendulum real-time Controller Implemented in Matlab-Simulink and LabView Environments* : bachelor's thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Control and Instrumentation, 2016. 47 p. Supervised by prof. Ing. Pavel Václavek, Ph.d.

DECLARATION

I declare that I have written my bachelor's thesis on the theme of "Inverted Pendulum real-time Controller Implemented in Matlab-Simulink and LabView Environments " independently, under the guidance of the bachelor's thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the bachelor's thesis I furthermore declare that, as regards the creation of this bachelor's thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

I would like to give special thanks to my supervisors at TUT Hannu Koivisto and Mikko Salmenperä for their patience, professional guidance, time, study materials and for providing me with the entry to the workroom; to my supervisor at BUT prof. Ing. Pavel Václavek, Ph.D. for his help and for allowing me to write the Bachelor's thesis abroad; to James Grant from NI support team for assisting me with the system setup and finally to my brother and family for their mental and financial support.

Brno ....

author's signature

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

In the last years, Network Control Systems (NCSs) have experienced a significant growth. They have been applied in the industrial control applications such as automobiles, aircraft, hospitals, domestic robots, monitoring or manufacturing plants. Connecting control system components via network can simplify the systems and reduce investments. Data might be shared efficiently and spread easily over a large physical space. Furthermore, unnecessary wiring is eliminated. It is easy to add some components to the system at low cost and without heavy structural modifications. On the other hand it introduces some constraints which we have to consider in designing the controller [1].

In this thesis the real-time network communication is developed and the rotational inverted pendulum (Futura pendulum) is controlled over the network. The goal is to keep the inverted pendulum in its upright position. First, the LabView program running on National Instrument CompactRIO reads the measurements of the two angles of the Futura pendulum system then the LabView software sends those values to Simulink via UDP protocol. In Simulink there is a linear discrete state space control design based on LQR with an output value representing a voltage which is returned back to the LabView program. Finally, LabView program sets the voltage to the DC motor which drives the pendulum's arm.

The thesis is organized as follows - In the first chapter the reader gets familiar with all the hardware used in the laboratory. Chapter two provides the reader with the information about the Hardware Setup. In chapter three I experiment with different means on how to send data over to the network and choose the best option. Finally, in chapter 4 I design the control that keeps the pendulum in its upright position and show the experiments with the certain controller.

The developed system will be used to demonstrate variable delay effects on control loop in various courses given by the Department of Automation Science and Engineering at Tampere University of Technology.

1 SYSTEM HARDWARE

In this chapter we get familiar with all the elements used in the laboratory. The main QUBE-Servo components are listed and characterized.

1.1 System scheme

There are various options on how to interface QUBE-Servo model with different I/O interfaces - the QUBE-Servo USB Interface, the QUBE-Servo myRIO Interface and the QUBE-Servo Direct I/O Interface which is the one used in my experiment. Only the QUBE-Servo interface includes a built-in Data Acquisition (DAQ) device and an integrated amplifier. The QUBE-Servo Direct I/O Interface has an amplifier but it does not have a built-in DAQ system. Instead, an external Q1-cRIO Quanser module is used for data acquisition which can easily interface with the Quanser QUBE-Servo.

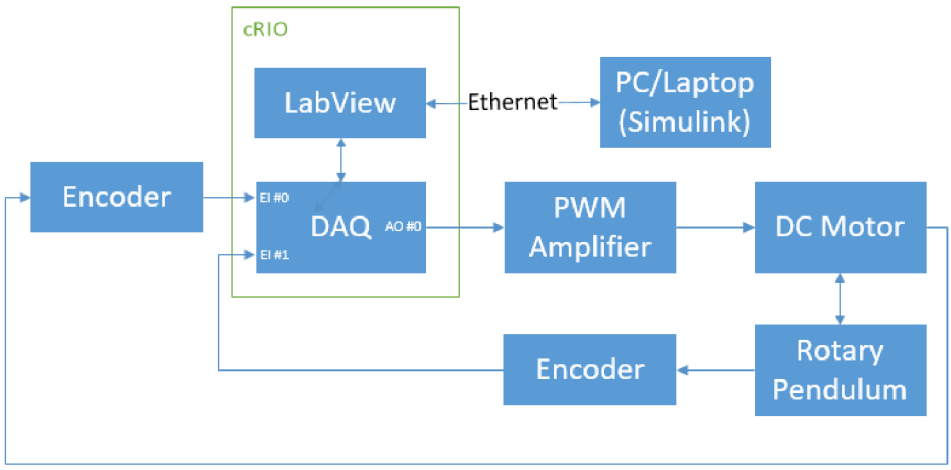


Figure 1.1: Interaction between QUBE-Servo components [2]

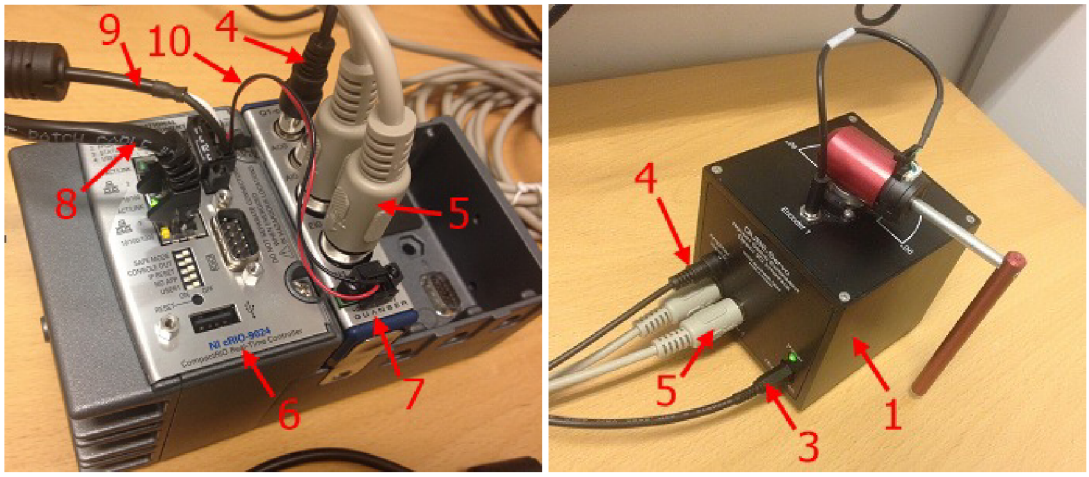
The diagram given in Fig. 1.1 illustrates the interaction between the QUBE-Servo components. The motor and the pendulum Encoders are connected to the Encoder Input (EI) channels #0 and #1 of the DAQ device. The power amplifier is connected to the Analog Output (AO) channel #0. The Direct Current (DC) motor is then driven by the power amplifier. The Q1-cRIO Quanser module is used as an external DAQ device which is a part of the NI CompactRIO. There is also LabView software running on the NI CompactRIO through which the data is measured and sent to another computer with Simulink via a UDP protocol. The controller made in Simulink returns the value representing the voltage and sends it back to the LabView software which sets the voltage to the physical system [2].

1.2 Hardware Elements

In Tab. 1.1 there are listed all the elements used for proper communication and function. The Fig. 1.2 depicts the wiring related to the cRIO-9024 and the QUBE-Servo module. The cRIO-9024 requires a 35 VDC external power supply [5]. The QUBE Servo is supplied with the power of 15 V 2 A.

Table 1.1: Hardware elements

ID	Component
1	QUBE-Servo (Direct I/O interface)
2	Rotary Pendulum (ROTPEN) module
3	15 V 2.0 A power supply
4	RCA cable
5	Two 5-pin-DIN cables
6	cRIO-9024
7	Q1-cRIO module
8	Ethernet cable
9	35 VDC power supply
10	Red and black power wires
11	Switch HP 1810-8G
12	PC/Laptop



(a) cRIO-9024 (b) QUBE-Servo

Figure 1.2: System elements

1.3 QUBE-Servo Hardware components

The QUBE-Servo hardware components are listed in Tab. 1.2.

Introduction of QUANSER

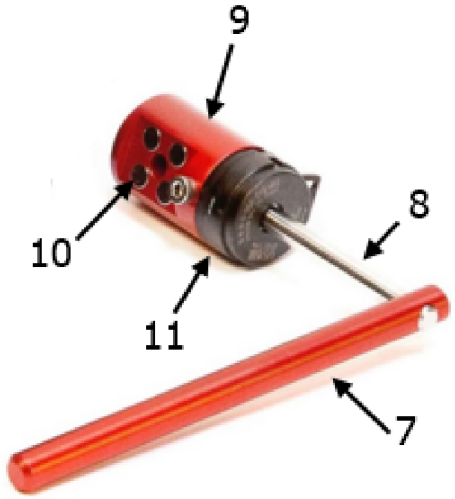
Quanser was set up in 1989 in response to the need in educational system and research for real-time control systems equipment. It became immediately popular among the universities around the world for its products control including specific applications such as unmanned vehicles, robotics, flight control and others.

In 2003 Quanser entered into the partnership with the National Instrument which led to the development of new series of devices. Quanser innovations proceed and nowadays it is regarded as being the World leader in education and research for control of design and implementation. Its products are used in over 2500 education and research institutions [3].

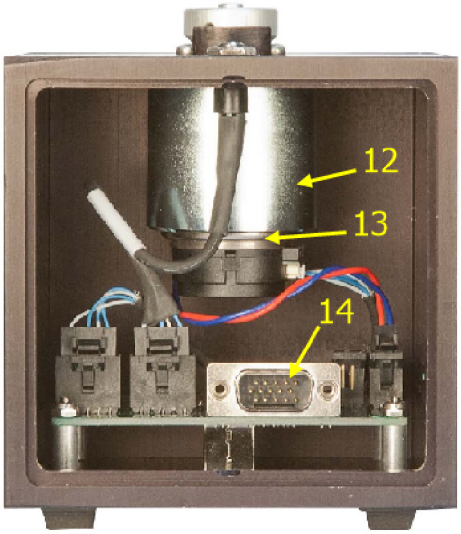
Table 1.2: QUBE-Servo components [2]

ID	**Component**	**ID**	**Component**
1	Aluminium chassis	10	Rotary pendulum magnets
2	Module connector	11	Pendulum encoder
3	Module connector magnets	12	DC Motor
4	Module encoder connector	13	Motor encoder
5	Power connector	14	QUBE-Servo DAQ/amplifier board
6	Power LED	15	Encoder 0 connector
7	Pendulum link	16	Encoder 1 connector
8	Rotary arm rod	17	Amplifier Input 0 connector
9	Rotary arm hub		

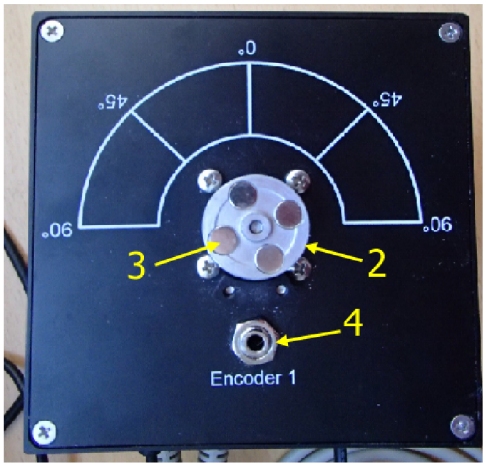
(a) QUBE-Servo DirectI/O Interface [2]



(b) QUBE-Servo Pendulum module [2]



(c) QUBE-Servo Interior [2]



(d) QUBE-Servo Top View

Figure 1.3: QUBE-Servo components

DC Motor

Inside the aluminium box there is a direct-drive 18 V brushed DC motor. The DC motor parameters are listed in Tab. 1.3.

Encoder

The Encoders used in the QUBE-Servo system are the US Digital E8P-512-118 single-ended optical shaft Encoders. They measure the angular position of the DC motor and the pendulum on the QUBE-Servo and convert them to digital code. Its

output is 2048 counts per revolution in a quadrature mode which means 512 lines per revolution [2].

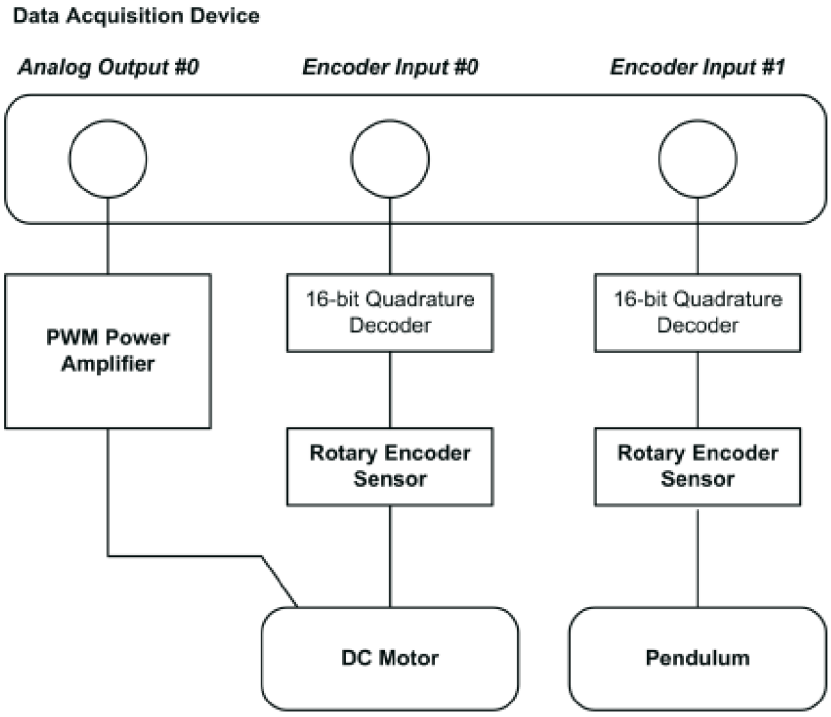


Figure 1.4: Interaction of the DAQ device with the QUBE-Servo components [2]

Power amplifier

The PWM voltage-controlled power amplifier situated on the QUBE-Servo circuit board provides 2 A peak current and 0.5 A continuous current. The output voltage is between ±10 V [2].

Data Acquisition (DAQ) Device - Quanser Q1-cRIO

The Quanser Q1-cRIO is the data acquisition and control module for the National Instrument CompactRIO controller. It provides one Analog Input (AI), one AO, and two single-ended Encoder input interfaces as shown in Fig. 1.5 which enable us to interface with the pendulum. All inputs and outputs are accessed simultaneously which allows for the real-time control. The Q1-cRIO requires the Quanser Rapid Prototyping (QCRP) toolkit for LabView. The Q1-cRIO is used with NI cRIO 9024 [4].

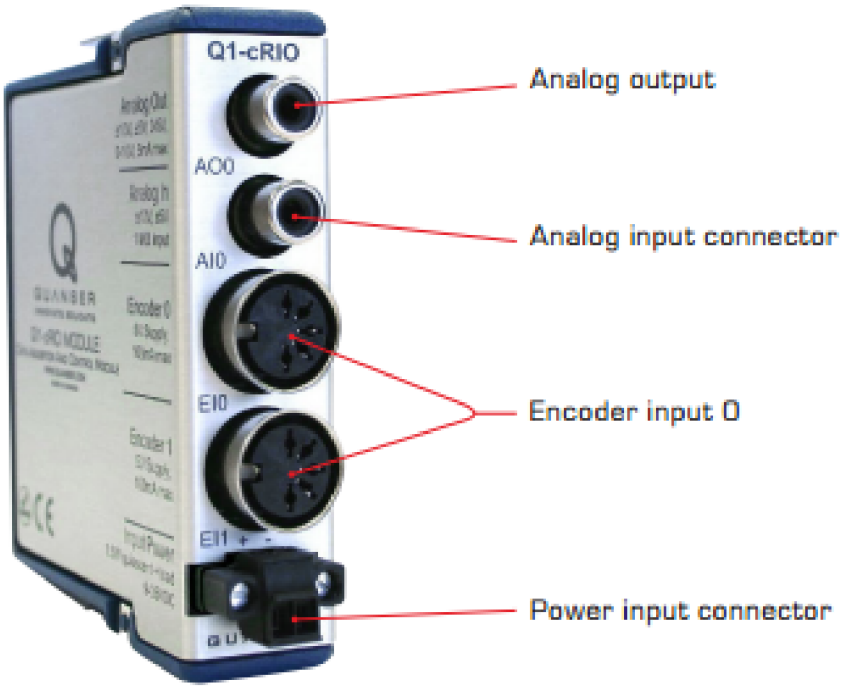


Figure 1.5: Q1-cRIO module

Amplifier Input Connector

The Amplifier Input RCA connector is single-ended with a range of ±10 V. As presented in Fig. 1.4, it is connected to the power amplifier which then drives the DC motor [2].

Encoder Connector

The Encoder connector pin-out is shown in Fig. 1.6. The Encoder 0 and Encoder 1 5-pin DIN connectors output the measurements from the DC motor Encoder and the pendulum module Encoder [2].

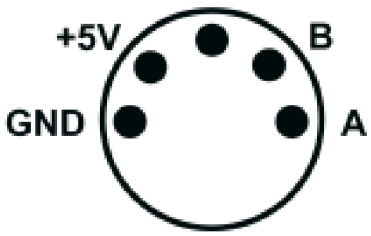


Figure 1.6: 5-pin DIN Encoder pin-out [2]

Table 1.3: Qube-Servo system parameters [2]

Symbol	Description	Value
DC Motor		
V_{nom}	Nominal input voltage	18.0 V
τ_{nom}	Nominal torque	22.0 mN·m
ω_{nom}	Nominal speed	3050 RPM
I_{nom}	Nominal current	0.540 A
R_m	Terminal resistance	8.4 Ω
k_t	Torque constant	0.042 N·m/A
k_m	Motor back-emf constant	0.042 V/(rad/s)
J_m	Rotor inertia	4.0 x 10^6 kg·m^2
L_m	Rotor inductance	1.16 mH
m_h	Module attachment hub mass	0.016 kg
r_h	Module attachment hub radius	0.0111 m
J_h	Module attachment moment of inertia	0.6 x 10^{-6} kg·m^2
Rotary Pendulum Module		
m_r	Rotary arm mass	0.095 kg
L_r	Rotary arm length (pivot to end of metal rod)	0.085 m
m_p	Pendulum link mass	0.024 kg
L_p	Pendulum link length	0.129 m
Motor and Pendulum Encoders		
	Encoder line count	512 lines/rev
	Encoder line count in quadrature	2048 lines/rev
	Encoder resolution (in quadrature)	0.176 deg/count
Amplifier		
	Amplifier type	PWM
	Peak current	2 A
	Continuous current	0.5 A
	Output voltage range	±10 V

2 SYSTEM SETUP

This section describes how to connect all elements in the network. The connection procedure is given below, the wiring is summarized in Tab. 2.1.
I followed these steps to connect all the elements:

1. After ensuring that power supply is switched off, I connected the primary power supply and the optional secondary supply to the power connector cRIO-9024.

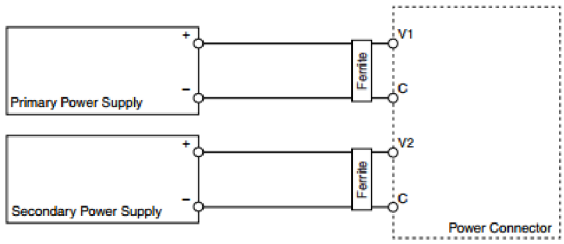


Figure 2.1: cRIO-9024 Power Connections [5]

2. I connected the cRIO-9024 to an Ethernet network using an Ethernet cable and an Ethernet port 1 [5].
3. After that I connected the Power input connectors on the Q1-cRIO module to the power connector cRIO-9024 with the red and black power wires (the possitive wire to the V1 terminal and the negative to one of the C terminals) [6].

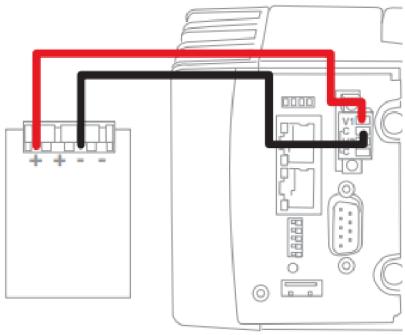


Figure 2.2: Connection of the cRIO-9024 and the Q1-cRIO module [6]

4. I connected the Analog Output #0 on the Q1-cRIO to the Amplifier Input 0 socket on the QUBE-Servo by using the RCA cable.
5. I connected the Encoder Input #0 on the Q1-cRIO module to the Encoder 0 of the QUBE-Servo using the 5-pin-DIN to 5-pin-DIN cable.
6. I attached the ROTPEN module to the motor hub using the magnets.

7. I connected the Encoder cable of the ROTPEN attachment to the Encoder 1 connector on the top of the aluminium box of the QUBE-Servo.
8. I connected the Encoder Input #1 on the Q1-cRIO module to the Encoder 1 of the QUBE-Servo using the 5-pin-DIN to 5-pin-DIN cable [2].

Table 2.1: QUBE-Servo wiring [2]

Cable	**From**	**To**	**Signal**
1	Q1-cRIO: Analog Output #0	QUBE-Servo Amplifier Input #0	Amplifier voltage driving command
2	Q1-cRIO: Encoder Input #0	QUBE-Servo Encoder #0 connector	Motor encoder measurement
3	Q1-cRIO: Encoder Input #1	QUBE-Servo Encoder #1 connector	Pendulum module encoder measurement

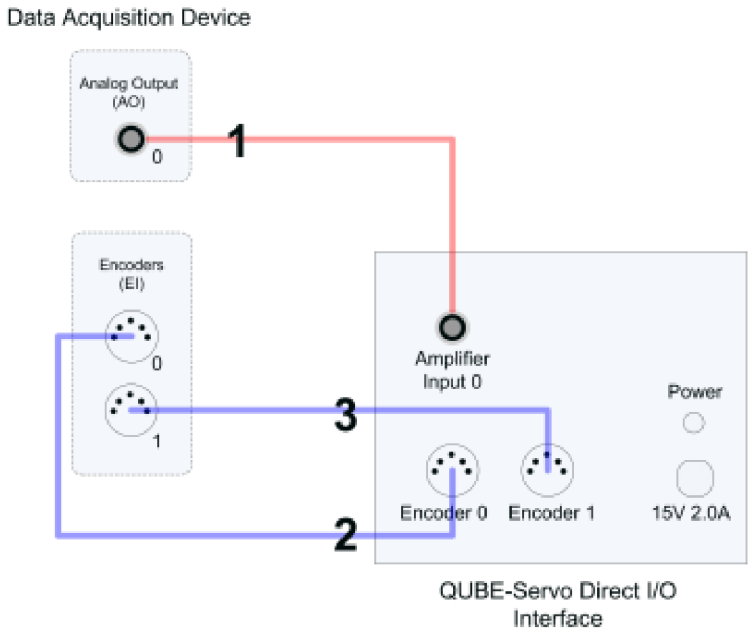


Figure 2.3: Connection between the QUBE-Servo and the Q1-cRIO module [2]

3 DATA EXCHANGE BETWEEN MATLAB SIMULINK AND LABVIEW

This section presents two methods (Data transfer over TCP/IP and UDP) for data exchange between the Simulink and the LabView. Another method is a Model Interface Toolkit (MIT) in the LabView - The simulink model is converted into a Dynamic Link Library (DLL) model and integrated in the LabView with MIT.

3.1 Data exchange over TCP/IP

TCP/IP is a set of communication protocols used on the Internet and other networks. One object typically acts as a server, and one or more objects act as clients. It is a connection-oriented service (two processes must first "handshake" with each other before sending away any packets) which provides reliable data transport (error correction). It is OS-independent and models that run on Simulink allow fast modifications. It also allows the distribution of computational effort, as Simulink can run in another computer.

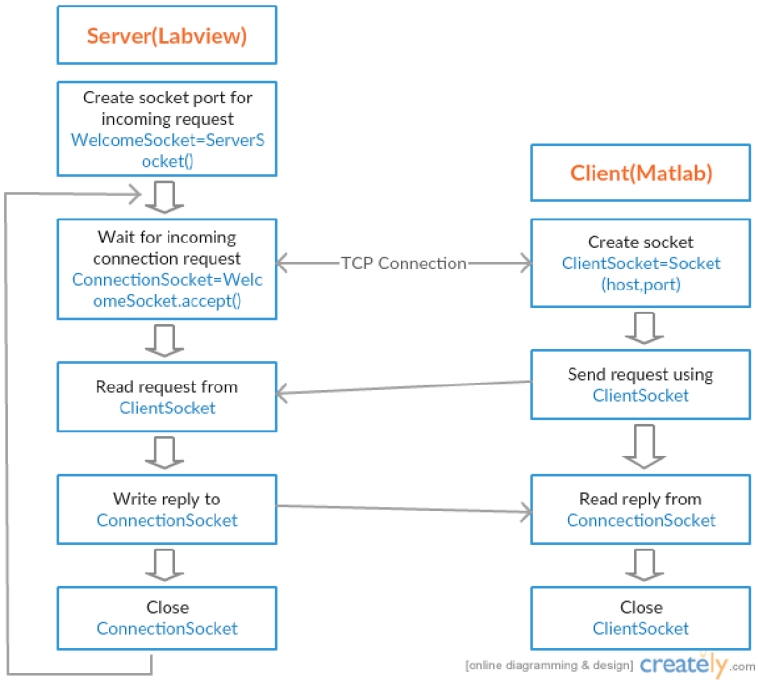


Figure 3.1: Data flow - TCP connection [7]

In this demonstration the LabView acts as a server and the Matlab acts as a client as shown in Fig. 3.1. First, the server socket (Welcome Socket-door) is created on the server and is waiting to be contacted by a client. In this case the port number

2057 identifies the process at the server. When a client sends a connection request to the server (knock on the door) the Connection Socket is created and TCP connection then establishes the virtual pipe between Connection Socket at the server and the Client Socket at the client. Now it is possible to exchange data between the Matlab and the LabView. In the diagram the data is sent from the LabView to the Matlab but it might also be done vice versa. At the end, the sockets are closed which also closes the TCP connection between the client and the server [7].

3.2 Data exchange via UDP

The UDP protocol can also be used to stream data from the Matlab to the LabView. It is a connectionless service (sends data without ever establishing a connection) which does not have any integrated error correction mechanisms (requires additional programming efforts). And like TCP, it is also OS-independent, we can easily modify the running model on Simulink and it can run on different computer. In my experiment I use a simple implementation - In Simulink there are UDP Send and UDP Receive blocks from the DSP System Toolbox [17]. In LabView the program is based on Simple UDP - Sender.vi.

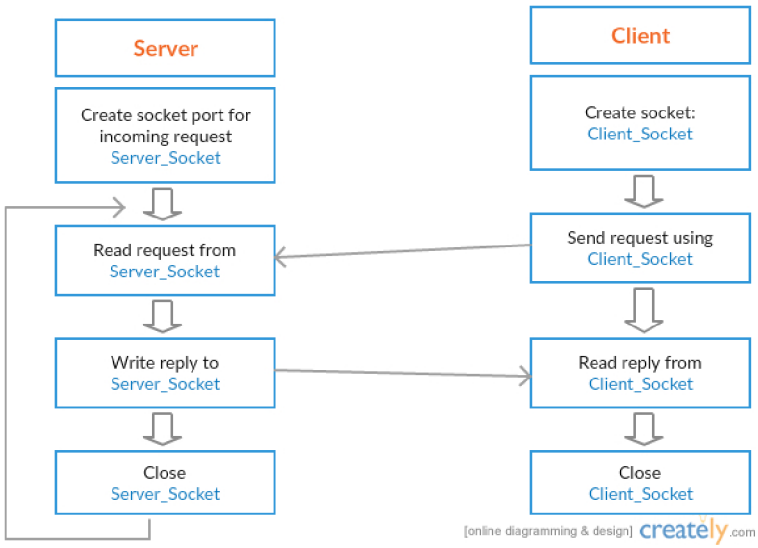
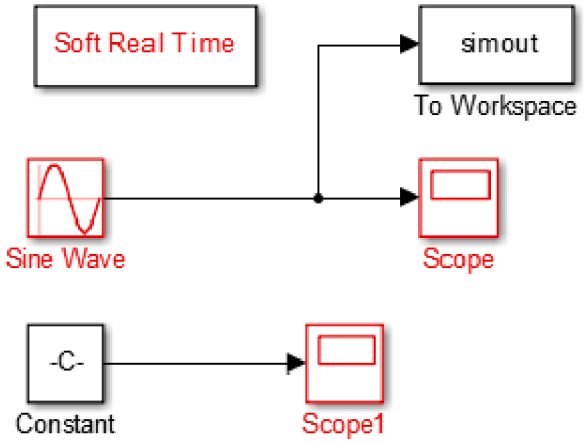


Figure 3.2: Data flow - UDP connection [7]

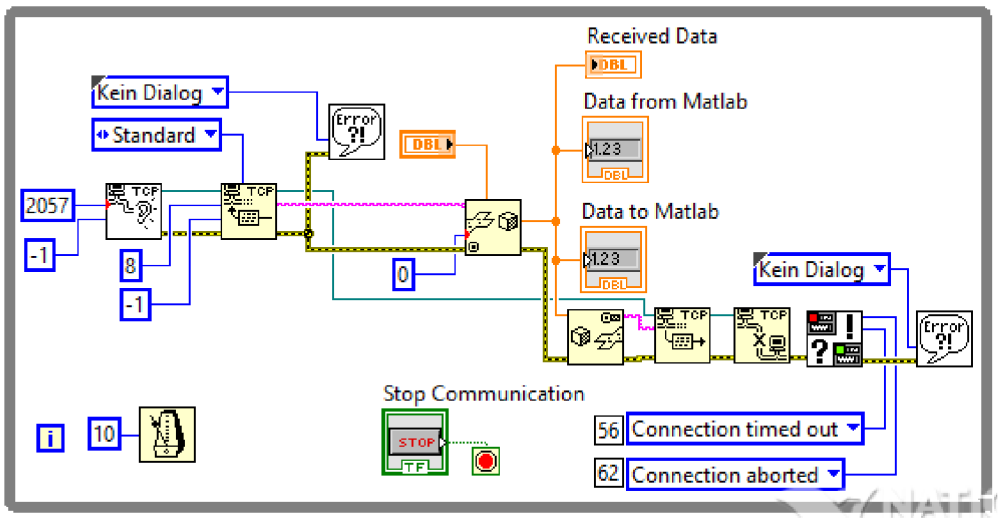
In this case, the Simulink and also the LabView can act as a server or a client. Depending on which port is listening. Nevertheless, both sides only send or receive packets from or to the other side. In contrast to TCP connection, there is no previous "handshaking". The example in Fig. 3.2 shows the data exchange from the Server to the Client using UDP [7].

3.3 The MATLAB and LabView data exchange over TCP/IP (running on the same host)

To exchange data over TCP/IP I used and modified the Matlab script and the LabView code from this webpage [8] - the code is presented and described in the appendix A and B. To run it in real-time it was necessary to add the Soft Real-Time Library [9]. Basically, the model generates the sine wave and sends the value to the Workspace. The block **Constant** includes a variable which comes from the LabView - Fig. 3.3a). For setting/getting parameters from/to the Simulink I used the functions `get_param`, `set_param` - appendix A.



(a) Simulink model

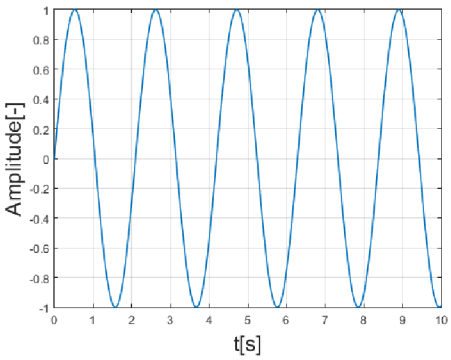


(b) LabView code

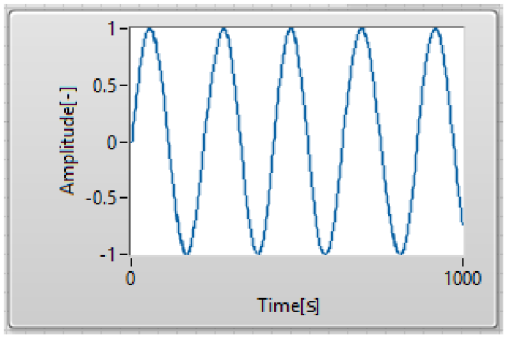
Figure 3.3: TCP/IP: the Simulink model and the LabView code

In the LabView, SubVI - Listen.vi is a sort of door that is waiting for the knock

from a client. When the client contacts the server the connection is established and we can read the data with the block **TCP Read**. The output of the **TCP Read** is a string (in our case a binary string because we send the type Double from the Matlab). I choose to convert from the binary string to the Double using the function **Unflatten From String**. For sending data to the Matlab there is a block **TCP Write** - it requires a string (in this case it is again a binary string). I need to use **Flatten To String** function which converts anything to a flattened string of binary values [10]. At the end the connection is closed with the **TCP Close Connection**.



(a) Data displayed in the Simulink



(b) Data displayed in the LabView

Figure 3.4: TCP/IP: Data exchange

The sampling rate is set to *10* ms. The data exchange demonstration is shown in Fig. 3.4.

The Fig. 3.5 depicts delta time between the received frames in the Matlab which are about *16* ms on average. The average is calculated and displayed every *100* ms. In the graph there can be found some peaks. They represent the unsuccessful (high peaks) or successful (low peaks) packets of delivery causing the increase and decrease in time.

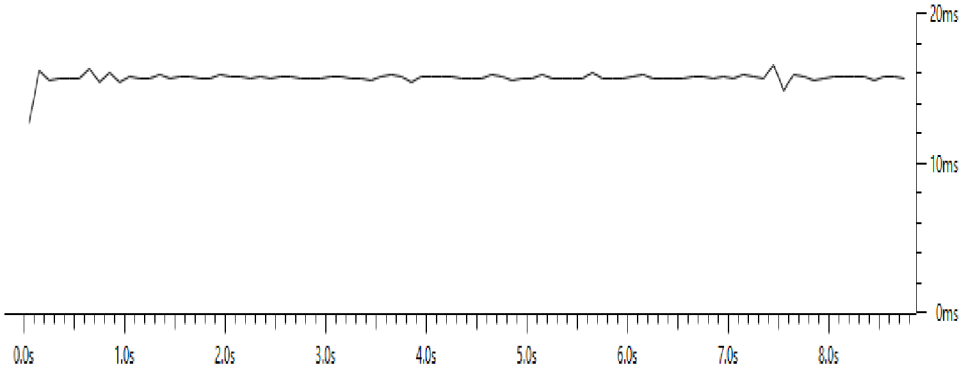
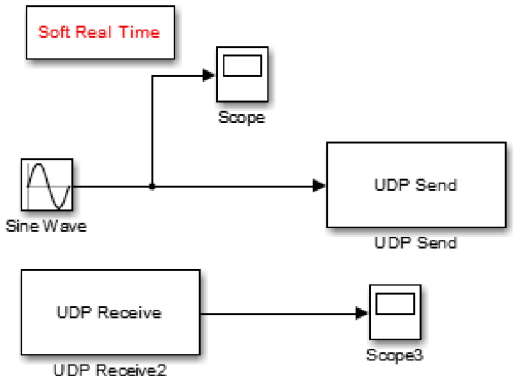


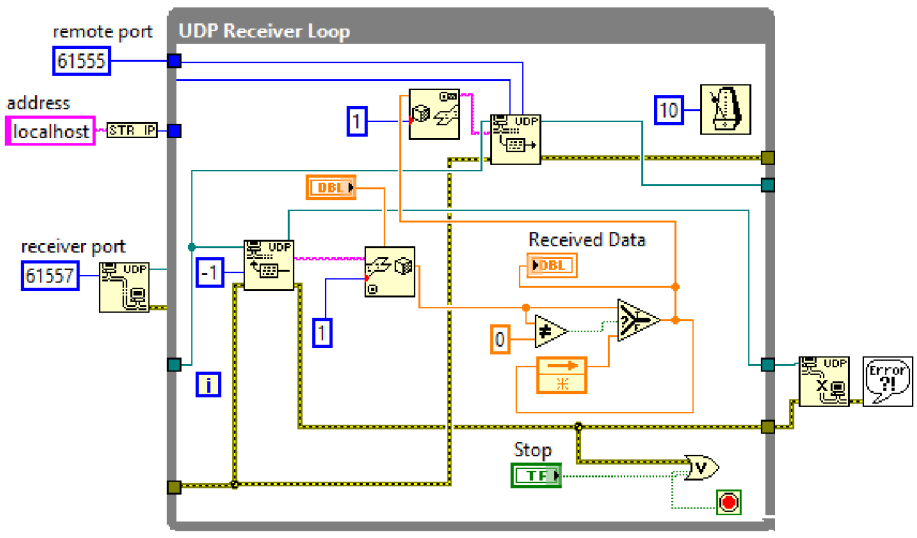
Figure 3.5: TCP: Delta time between the received packets

3.4 The MATLAB and LabView data exchange via UDP (running on the same host)

In Simulink the data is sent and received by the blocks **UDP Send** and **UDP Receive** from the DSP System Toolbox [17]. These blocks also exist in the Instrument Control Toolbox [18]- The main difference is that the blocks from the DSP System Toolbox are suitable only for one-dimensional vectors whereas the blocks from Instrument Control Toolbox can be used for multi-dimensional vectors. In this application I wanted to send and receive only the one-dimensional vectors so I used the blocks from the DSP System Toolbox. The blocks from the Instrument Control toolbox will be used later in section 3.5. To run the model in real-time I used the Soft Real-Time Library again. The model is shown in Fig. 3.6a).



(a) Simulink model

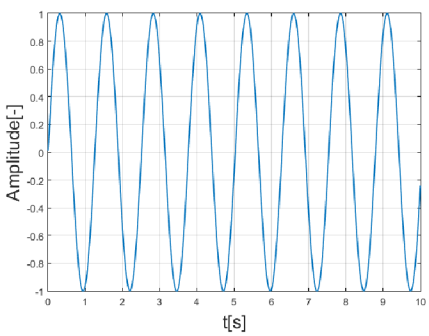


(b) LabView code

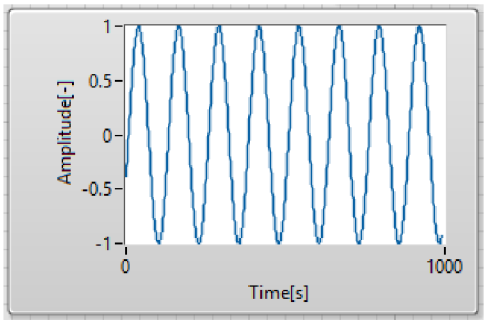
Figure 3.6: UDP: the Simulink model and the LabView code

In the LabView the program is based on examples of a Simple UDP - Sender.vi and Simple UDP Receiver.vi. Initially the connection is established by the **UDP Open**. To read the data there is a block named **UDP Read**. And like in the TCP Connection, the output from the **UDP Read** is a string and that is the reason why we need to use the **Unflatten From String**. The code in the LabView (Fig. 3.6b) does not keep the data all the time - and because of that it goes down to zero among the received values. We need to check whether the received value is zero or not. If so, the previous value is used. Next, to send data to the Simulink there is the block named **UDP Write**. To send anything via UDP, the block **Flatten To String** is necessary again. Finally, the connection is closed by the **UDP Closed**.

Sample time is set to *10* ms. This speed of data exchange should be enough to control the inverted pendulum. In Fig. 3.7 we can see that the data is exchanged continuously, without any breaks on both sides.



(a) Data displayed in the Simulink



(b) Data displayed in the LabView

Figure 3.7: UDP: Data exchange

The Fig. 3.8 depicts delta time between the received frames in the Matlab which is about *11* ms on average. To compare Fig. 3.8 with Fig. 3.5, delta time between the received packets sent via UDP does not fluctuate as much as over TCP/IP.

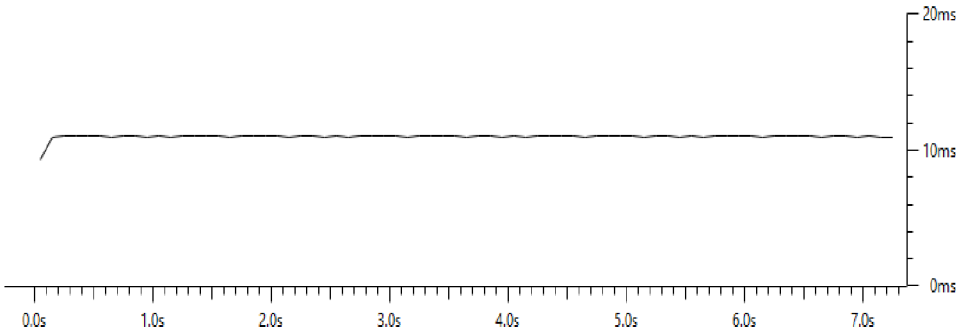


Figure 3.8: UDP: Delta time between the received packets

3.5 Data exchange applied to the actual system

I analysed data transfer from the Matlab to the LabView and vice versa over the TCP/IP and UDP. Both of them met the requirements. My aim was to study how fast the data transfer could be. The Round Trip Time should be reached in about *10* ms. In the TCP/IP connection I achieved the time difference of *t=16* ms between the received packets. In the UDP connection I achieved a more desirable result of *t=11* ms which is closer to *10* ms than in the TCP/IP connection. For the remote control purposes the UDP connection is more appropriate. It does not have an error correction, it is less complicated and it is faster than the TCP/IP. We do not need to receive all the packets when controlling the pendulum. If some packets are dropped it is useless to resend them again - we need an actual value of the pendulum position. An effort of the TCP/IP connection to resend the packet again and again might cause the fact that the value will not be actual and the pole may fall down.

When the data transfer is applied to the actual system, the situation becomes more complicated. The primary problem is that there are two kinds of measurements (the angle of the pole, and the angle of the arm) which must be sent from the LabView to the Matlab. It means that we need to send a multi-dimensional vector instead of one-dimensional. The second issue is that the data transfer does not run on the same host but it runs on two devices with different IP addresses. As a result, we can get a greater delay in communication which we have to take into consideration during the design of the control. In this section I am trying to deal with transferring of two numbers from the LabView to the Matlab and I am trying to measure the delay important for the design of control.

In the following application I display values, coming from the LabView, in the Simulink. In the Simulink I generated the square signal representing the voltage and sent it to the LabView. The pole was in its downward position. This application assured me that I have a proper communication, necessary for the control of the inverted pendulum. Now I am going to demonstrate how the LabView software differs from the one in section 3.4. The code is presented in appendix C.

As the first step I will specify the board type in the **HIL Initialize** block. We work with q1_single. **CL HIL Read** block reads the counts from the Encoder and then **Counts to Angles** calculates certain angles in radians from the counts. Before sending the data to the Matlab I need to remove the first 4 bytes because they represent the header which contains information about the packets. It is done by the **String Subset**. In the end, the data coming from Matlab is written on the board with the **CL HIL Write** [11]. The rest of the code is the same as it was

previously described in section 3.4.

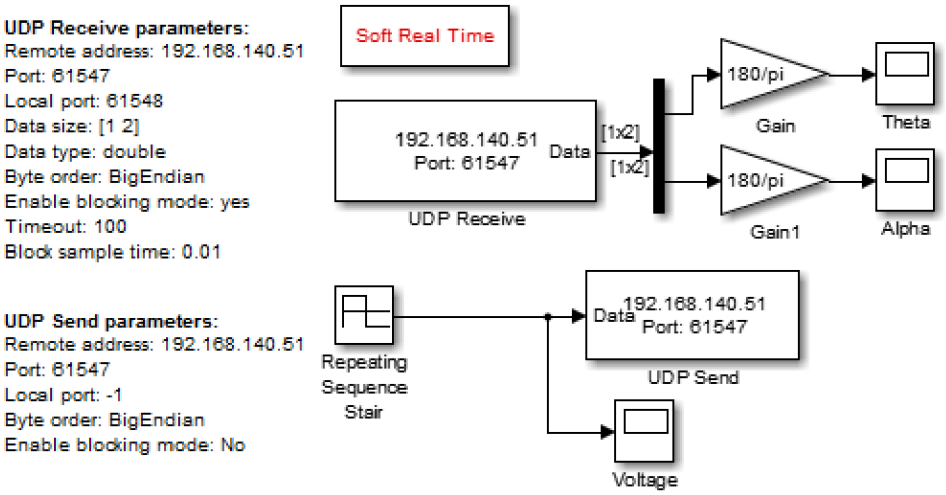


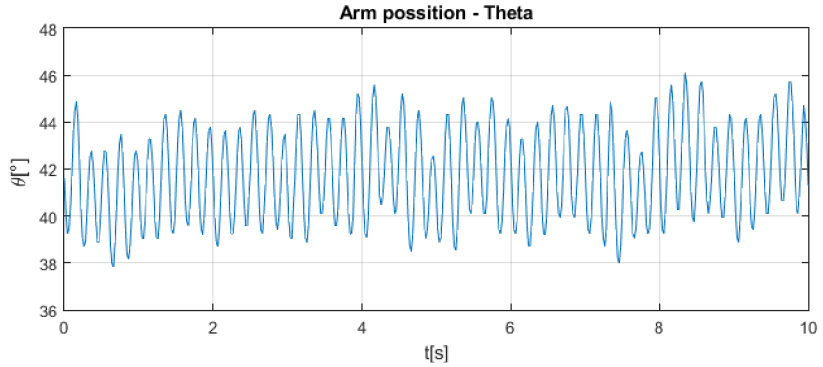
Figure 3.9: UDP: Simulink model

In the Simulink the data is sent and received by the blocks **UDP Send** and **UDP Receive** from the Instrument Control Toolbox. The **UDP Receive** enable us to receive a multi-dimensional vector as shown in Fig. 3.9 - parameter `Data size`. After that the square wave signal representing voltage is generated and sent to the LabView.

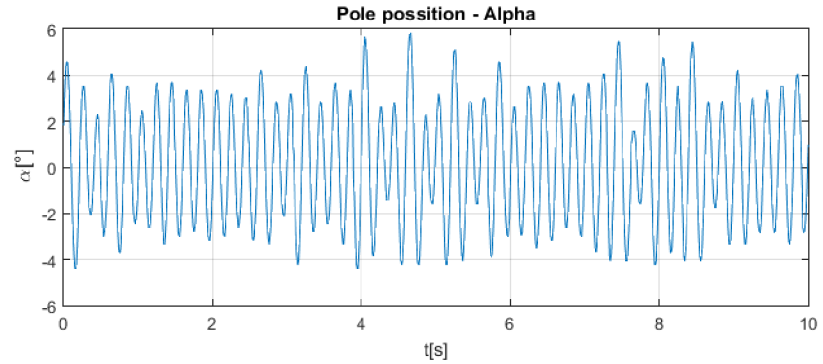
In the LabView the sampling time is set to *1* ms, in the Simulink it is *10* ms. In the LabView the sampling time is much shorter to avoid possible consequences of an unsuccessful measurement in the LabView. Basically, if one of the 10 measurements in the LabView is done correctly then we reach *10* ms sampling time in the Simulink.

The result of the data transfer is shown in Fig. 3.10. It can be observed that the behaviour of the angles is not particularly symmetrical (theoretically, it should be symmetrical because it generates square signal). It might be caused by the delay. If there is a delay in the generated signal in the Simulink then in the LabView the previous value is used which causes that the positive or negative signal affects the motor for a longer time. Nevertheless, the position does not matter, in this case the most important thing is that the data is transferred successfully.

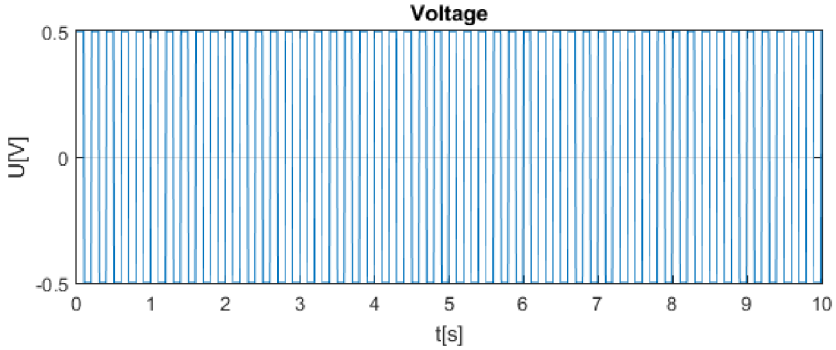
The Fig. 3.11 displays the delta time between the received packets. It is about *15* ms on average. There are some peaks which is a signal that some data is delivered some of it is lost. As a result there is this fluctuation. The controller dealing with the balance will be designed based on an average time delay.



(a) Theta



(b) Alpha



(c) Voltage

Figure 3.10: Data displayed in the Simulink (pole in its downward position)

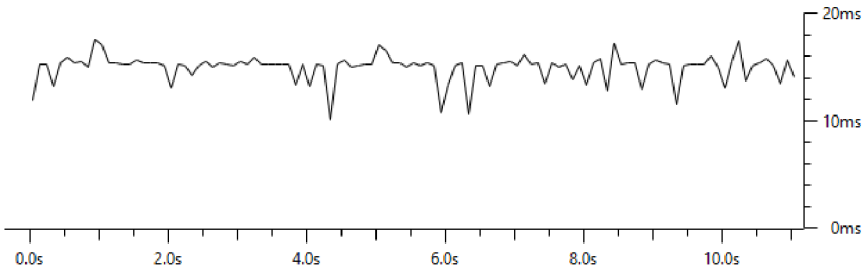


Figure 3.11: Delta time between reached packets

4 DESIGN OF THE CONTROL

4.1 Control Law

All linear dynamic systems can be described by the differential equations. Those differential equations of any order can be transformed into a set of first order equations called state-space description. The general state-space description of a continuous system is expressed as

$$\begin{aligned} \dot{x} &= Fx + Gu, \\ y &= Hx + Ju, \end{aligned} \tag{4.1}$$

where **x** is a column vector called the **state** containing n elements for an nth-order system, **u** is the $m \times 1$ input vector of the system, **y** is the $p \times 1$ output vector, **F** is an $n \times n$ system matrix, **G** is an $n \times m$ input matrix, **H** is a $p \times n$ output matrix, and **J** is $p \times m$[1].

I applied the control from the computer through a ZOH. Therefore, the equations have a discrete representation as

$$\begin{aligned} x(k+1) &= \phi x(k) + \Gamma u(k), \\ y(k) &= Hx(k) + Ju(k), \end{aligned} \tag{4.2}$$

where

$$\begin{aligned} \phi &= e^{FT}, \\ \Gamma &= \int_0^T e^{F\tau} d\tau G. \end{aligned} \tag{4.3}$$

In theMatlab we can easily transform the continuous system to the discrete one with a sample period T, using the Matlab Control System Toolbox (CST)

$$sysD = c2d(sysC, T). \tag{4.4}$$

The state-space methods consist of two independent steps. The first one is the control law and it assumes that we have all the states available for feedback purposes. The other step is to design an estimator (aka observer), which estimates the entire state vector based on measurements of the rest of the known states. The final control design consists of the estimator and the control law based on the estimator states rather than on the actual states [12]. In our design the estimator is not taken into

[1] Franklin, Gene F., J. David Powell and Michael Workman. Digital control of dynamic systems. Page 12. [12]. It is also common to use **A**, **B**, **C**, **D** instead of **F**, **G**, **H**, **J**.

account. We do not need it because two of the states are measured and the two of them are calculated with a filter (we know all the states).

The control law is simply the feedback of all the state elements multiplied by the gain **K**.

$$u = -Kx = -\begin{bmatrix} K_1 & K_2 & \dots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}. \tag{4.5}$$

It can be observed that this structure does not allow to set the reference input. The control law described in Eq. 4.5 assumes that $r = 0$ and that is the reason why it is usually referred to as a regulator. To introduce the reference input $r \neq 0$ requires further studies that are not necessary for my control [12].

Substituting Eq. 4.5 in Eq. 4.2, I get

$$x(k+1) = \phi x(k) - \Gamma K x(k). \tag{4.6}$$

The z-transform of Eq. 4.6

$$(zI - \phi + \Gamma K)X(z) = 0. \tag{4.7}$$

At the end, the characteristic equation of the closed loop system with the control law looks as follows

$$|\ (zI - \phi + \Gamma K)\ | = 0. \tag{4.8}$$

4.2 State-Space Model for Systems with Delay

In this section I would like to present the discrete state models including a time delay in the model[2]. The continuous state-space model including the delay is

$$\begin{aligned} \dot{x} &= Fx(t) + Gu(t - \lambda), \\ y &= Hx. \end{aligned} \tag{4.9}$$

If we separate the system delay λ into an integer number representing the sampling periods plus a fraction, we can define an integer l and a number m as

$$\begin{aligned} &\lambda = lT - mT, \\ &l \geq 0, \\ &0 \leq m \leq 1. \end{aligned} \tag{4.10}$$

[2]Only the main steps are discussed. The formulas are derived step by step in details in the book Franklin, Gene F., J. David Powell and Michael Workman. Digital control of dynamic systems. Pages 110-114. [12]

With this substitution, the discrete system is described as

$$x(kT+T) = \phi x(kT) + \Gamma_1 u(kT - lT) + \Gamma_2 u(kT - lT + T). \tag{4.11}$$

where

$$\phi = e^{FT}, \qquad \Gamma_1 = \int_{mT}^{T} e^{F\eta} G d\eta, \qquad \Gamma_2 = \int_{0}^{mT} e^{F\eta} G d\eta. \tag{4.12}$$

If we consider $l > 1$, the equations are

$$x(k+1) = \phi x(k) + \Gamma_1 u(k-l) + \Gamma_2 u(k-l+1). \tag{4.13}$$

Now it is necessary to eliminate the past controls up to *u(k)*. We introduce l new variables such that

$$x_{n+1}(k) = u(k-l), \qquad x_{n+2}(k) = u(k-l+1), \qquad x_{n+l}(k) = u(k-1). \tag{4.14}$$

It results in increasing the dimension of matrices depending on the delay. The final structure of the system looks like this

$$\begin{bmatrix} x(k+1) \\ x_{n+1}(k+1) \\ x_{n+2}(k+1) \\ x_{n+3}(k+1) \\ \vdots \\ x_{n+l}(k+1) \end{bmatrix} = \begin{bmatrix} \phi & \Gamma_1 & \Gamma_2 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ x_{n+1}(k) \\ x_{n+2}(k) \\ x_{n+3}(k) \\ \vdots \\ x_{n+l}(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} u(k),$$
$$y(k) = \begin{bmatrix} H & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x(k) \\ x_{n+1}(k) \\ \vdots \\ x_{n+l}(k) \end{bmatrix}. \tag{4.15}$$

In the Matlab we can discretize the continuous state space model with CST

$$[Phi, Gam, Cd] = c2dt(A, B, C, T, d), \tag{4.16}$$

where **A**, **B**, **C** are the matrices describing the continuous time system, T represents the sampling time and d is a time delay.

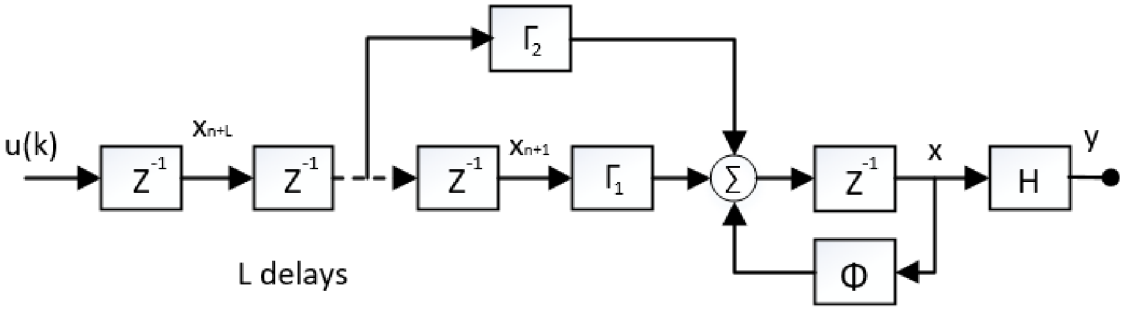


Figure 4.1: System with a delay of more than one period [12]

4.3 LQR Steady-State Optimal Control

One of the possibilities to stabilize the inverted pendulum is by optimal control methods which are attractive because they can easily deal with MIMO systems. There are also very efficient computation tools which help designer to find the proper feedback gain **K** [12]. For a discrete plant

$$x(k+1) = \phi x(k) + \Gamma u(k), \tag{4.17}$$

we look for a way to find the State-Variable Feedback (SVFB) control

$$u = -Kx \tag{4.18}$$

that minimize the **cost function**

$$J(x_k) = \frac{1}{2}\sum_{k=i}^{\infty}(x_i^T Q x_i + u_i^T R u_i) \tag{4.19}$$

with design symmetric matrices $Q = Q^T \geq 0$, $R = R^T \geq 0$ based on the relative importance of the certain states and controls. Matrices must be nonnegative definite[3], which is easily accomplished by choosing matrices to be diagonal with all numbers on the main diagonal positive or zero. **Q** is an $n \times n$ matrix and **R** is an $m \times m$ matrix where m is number of control inputs. For the SISO systems and also in this case **R** is a scalar. The weights of those matrices are picked by trial-and-error method. This method is known as the Linear Quadratic Regulator (LQR), since the system is linear and the cost is quadratic [12].

Substituting SVFB into Eq. 4.19 and after taking many other steps - described

[3]It ensures that $x^T Q x$ and $u^T R u$

step by step here [13], we define the optimal control gain as

$$K = (R + B^T PB)^{-1} B^T PA. \tag{4.20}$$

To find **P**, we solve the **Riccati equation** for **P**

$$A^T PA - P + Q - A^T PB(R + B^T PB)^{-1} B^T PA = 0. \tag{4.21}$$

Finally, we verify the performance. If it is not satisfactory we try again for different **Q**, **R**.

In the Matlab, there is CST for calculation optimal gain **K**

$$[K, S, e] = dlqr(A, B, Q, R). \tag{4.22}$$

In addition, it returns **S** of the associated discrete-time Riccati equation and closed-loop eigenvalues e = *eig(A-B*K)* [14].

4.4 Controllability

The procedure before designing the control for a certain system is the verification of its controllability[4]. If the matrix **C** described as

$$C = \begin{bmatrix} \Gamma & \Phi\Gamma & \dots & \Phi^{n-1}\Gamma \end{bmatrix} \tag{4.23}$$

is nonsingular, then we can convert the model to the canonical form and construct a control law. The system is controllable provided the rank r of the matrix **C** is n where n is a number of states [12].

It is possible to calculate in the Matlab by

$$\begin{aligned} Co &= ctrb(sys), \\ r &= rank(Co). \end{aligned} \tag{4.24}$$

[4]Further discussed in the book Franklin, Gene F., J. David Powell and Michael Workman. Digital control of dynamic systems. Pages 345-351.

4.5 Design of the Control for the actual system

From the Quanser Student workbook [15] I got the description of the linearized continuous-time state space model

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 149.226 & -0.0104 & 0 \\ 0 & 261.525 & -0.0103 & 0 \end{bmatrix} B = \begin{bmatrix} 0 \\ 0 \\ 41.718 \\ 49.133 \end{bmatrix}, \tag{4.25}$$

in which the state vector x is defined as

$$x = \begin{bmatrix} \theta & \alpha & \dot{\theta} & \dot{\alpha} \end{bmatrix}^T, \tag{4.26}$$

where θ is the angle of the arm, α is the angle of the pole and $\dot{\theta}$, $\dot{\alpha}$ are their velocities.

In section 3.5 I studied the delay in data transport. As it is presented in Fig. 3.11 the delay is around *d=15* ms which is the key factor in the control design. To discrete the continuous-time model I used the Matlab CST

$$[Phi, Gam, Cd] = c2dt(A, B, C, T, d), \tag{4.27}$$

where T represents the sampling period. I experimentally studied and got the best control results with the sampling period *T=0.01* s. Then I got the discrete-time model

$$\phi = \begin{bmatrix} 1 & 0.0075 & 0.01 & 0 & 0.0019 & 0.0006 \\ 0 & 1.0131 & 0 & 0.01 & 0.0018 & 0.0006 \\ 0 & 1.4987 & 0.9999 & 0.0075 & 0.2496 & 0.2487 \\ 0 & 2.6266 & -0.0001 & 1.0131 & 0.2475 & 0.2459 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \Gamma = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}. \tag{4.28}$$

It can be noticed that the dimension of the matrices increased by two. It is because *delay* $= 1.5 \times$ *sampling period*. If we compare system matrix ϕ in Eq. 4.28 and system matrix in Eq. 4.15 we could see that the structures of the matrices are the same. Even the elements in the matrix Γ_2 are non-zeros because the time delay is not a multiple integer of the sampling period - there is a fraction.

Then I wanted to verify whether the conversion from the continuous time model to the discrete time model maintains the physical meaning of the states. It might be made by temporarily defining velocities as outputs *C=eye(4)*, and applying CST *c2dt* (4.27).

The output matrix is then

$$Cd = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{4.29}$$

I came to a conclusion that $C = Cd$ which is a proof that the states have the same physical meaning in both representations which confirms that I do not need any estimator.

Now there is a discrete model of the actual system with a network delay. Before I design a controller I first verify that the system is controllable. As it was discussed in section 4.4 since the controllability matrix is 6×6, the rank of the matrix must be 6. For that purpose I used a Matlab CST

$$\begin{aligned} C &= ctrb(Phi, Gam), \\ n &= rank(C). \end{aligned} \tag{4.30}$$

An expected result was achieved *n=6*.

We verified that the system is controllable and I should be able to design a controller achieving the specific requirements. For a discrete Linear Quadratic Regulator (LQR) controller I can apply a Matlab CST

$$[K, S, e] = dlqr(A, B, Q, R). \tag{4.31}$$

The matrices **Q** and **R** are picked by trial-and-error method which means that I set the weights on the matrix elements and run the application. According to the system behaviour I tuned the weights and tried again. This process was repeated until I reached the desirable result.

The best control result was reached with matrices

$$Q = \begin{bmatrix} 50 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix} R = 1. \tag{4.32}$$

I put more weight on the previous inputs and the arm position. The more the state is penalized the more it influences the system behaviour [12]. As a result I got the

required control gain

$$K = \begin{bmatrix} -1.2329 & 22.221 & -0.5585 & 1.6634 & 0.2573 & 0.4454 \end{bmatrix}. \tag{4.33}$$

The Fig. 4.2 shows how the controller is wired in the Simulink. We have two

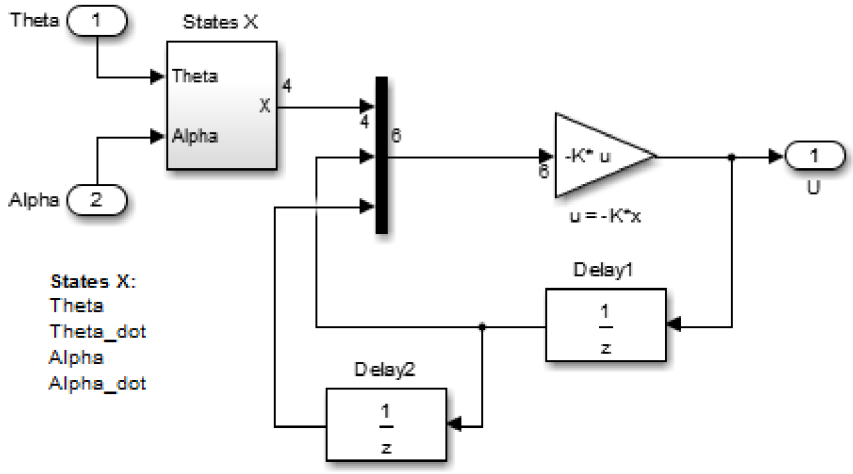


Figure 4.2: Controller wired in Simulink

measurements θ and α coming from the LabView. In the subsystem **States X** there are high pass filters $50s/(s+50)$ used to compute velocities $\dot{\theta}$ and $\dot{\alpha}$. Although I have a discrete time controller I can use continuous time filters because the sampling time is short enough which means that the discrete time signal is close to a continuous time behaviour. The states are multiplied by the calculated control gain **K** (4.33). The control output is a voltage U which drives the pendulum's arm.

The whole Simulink model is shown in Fig. 4.3. The data transfer is done in the same way as it was presented in Fig. 3.9. The **Enable Balance Control Switch** ensures that the control output is used if $\mid \alpha \mid \leq 10deg$, otherwise zero is returned.

The LabView code is made in the same way as shown in appendix C.

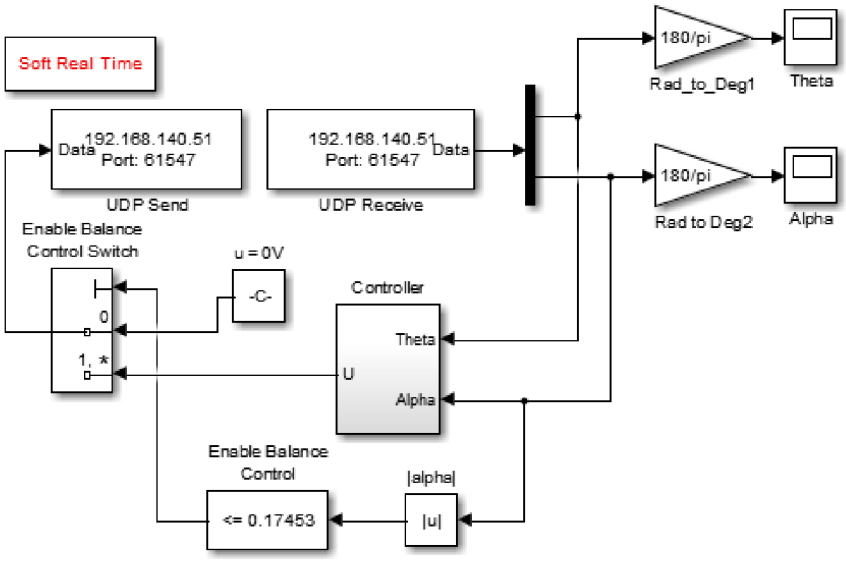


Figure 4.3: The Simulink model with LQR controller

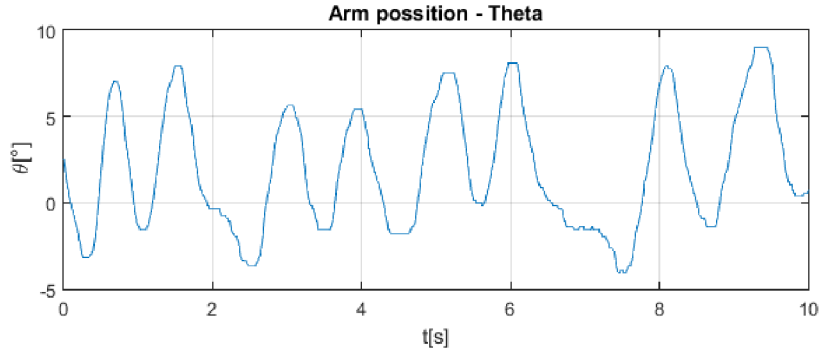
4.6 Experiments with the designed controller

In this chapter the experiments with the designed controller are shown.

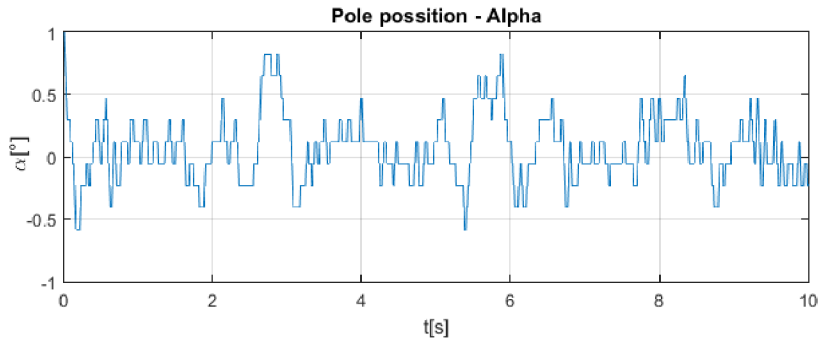
The first experiment confirmed that the controller works properly. I manually set the pole in its upright position and observed the arm's and pole's behaviour. In this experiment the controller just keeps the pendulum upright and there are no disturbances affecting the pendulum's position.

From the Fig. 4.4 it is clear that the pendulum stays upright and does not fall down. The pole moves in the range of $\pm 1°$ which is sufficient enough for the balance and I might have observed smooth movements of the arm and the pole while performing the experiment in the laboratory. Ideally (without delay and disturbances) the behaviour around the zero point would be periodic and symetrical. In this case it is not periodic because of variable delay.

Most of the signal which represents the arm position is above zero which means it does not fluctuate around zero (it seems that zero point is not set to *0* but to ± 2). This strange performance may be caused by an imperfect controller (when designing the control I put more emphasis on keeping the pole upright). Another reason why it performs in such a way could be - when the application is started, the position of the pole is $180°$ (no matter if the pole is downwards or upwards). The pole might have been swinging a little when the application was running and the position of $180°$ might have shifted which would also mean shift $0°$ in the upright position.



(a) Theta



(b) Alpha

Figure 4.4: Behaviour of stable inverted pendulum

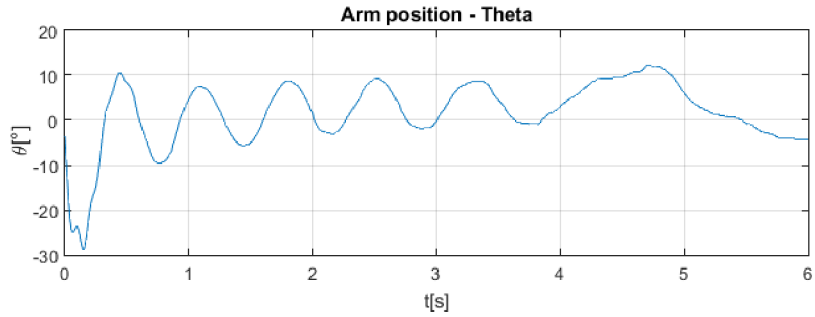
In the second experiment it is examined how the controller deals with disturbances. The disturbance is generated manually by a slight nudge to the pole. It is shown in Tab. 4.1 that the generated disturbance causes the peak value *-8.612°* which almost reaches the controlled limit $\pm 10°$.

Table 4.1: Information about the disturbance

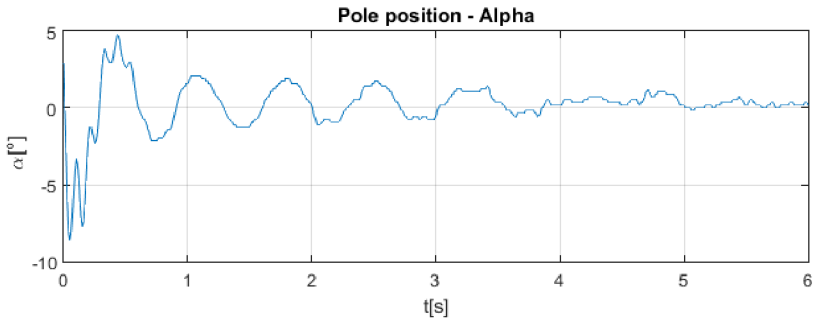
```
    RiseTime: 0.0071
SettlingTime: 5.7700
 SettlingMin: -8.6120
 SettlingMax: 4.7562
   Overshoot: 2.5006e+03
  Undershoot: 4.7088e+03
        Peak: 8.6120
    PeakTime: 0.0500
```

The Fig. 4.5 depicts the response to the disturbance. In Tab. 4.1 we can also see that the settling time is *5.77* s - this value is not really true because 2 % of *8.612°* (peak value) is *0.172°* and the settled position of the pole fluctuates in the range of $\pm 1°$ (as I found out in the first experiment) which means that if I display it for a longer time, the settle time in the Tab. 4.1 would probably increase. The pole position settles in the range of $\pm 1°$ at around *t=3* s.

Concerning the arm position, at the beginning there is a huge overshoot reaching almost *-30°*, which represents quick movement of the arm in order to balance the disturbance. Then the behaviour looks similar to the one in the first experiment. In the time interval from *t=4* s to *t=6* s the signal is not so steep as before. The packets are probably delivered successfully with small delays, the controller in Matlab receives values with minor differences and the final behaviour is more smooth. In the graph representing the pole position we may notice that the signal fluctuates really close to zero without any significant peaks.



(a) Theta



(b) Alpha

Figure 4.5: Dealing with disturbance

5 CONCLUSION

The goal of this thesis was to find the best solution to the problem of how to interconnect the Matlab Simulink and the Labview software and reach the shortest possible time delay between the received packets. Afterwards, I was supposed to design the controller which deals with this time delay and stabilizes the pendulum in its upright position.

All the Hardware used in the application including the parameters and specifications is described in the first chapter. The following chapter shows the instructions of the Hardware Setup and basic wiring.

In chapter three, I compared the TCP/IPand UDP protocols. As it was already mentioned in chapter 3.5 for data transfer we reached better results with the UDP protocol. It is less complicated, faster then TCP/IP and it does not have any error correction. After applying this data transfer application to the actual system I reached a time delay of *t=15* ms as is presented in Fig. 3.11.

In the chapter four I introduced the Control Law, Controllability and the State-Space Model for Systems with Delay - I mentioned how dimensions of the matrices change depending on the time delay and then, LQR Steady-State Optimal Control was discussed. All those discussed topics were utilized for designing the LQR control for the actual system. For assembling the control design I used the model description provided by the Quanser, I worked with the sampling period *T=0.01* s and I considered the time delay *t=15* ms which was studied in section 3.5. After fiddling with the parameters of the weighting matrix $\mathbf{Q}$ (4.32) I managed to keep the pendulum upright with the control gain $\mathbf{K}$ (4.33). I did not need any additional estimators because I measured and calculated all necessary states. The estimator would be useful if some states were not possible to measure or if there was some noise. Finally, I presented the experiments with the designed controller proving that the remote control works properly.

All tasks were successfully completed although it was still possible to improve the balance properties for instance to develop a more complex controller or to implement the controller in another environment. To process the Simulink model is time consuming. If I built the stand alone application written in a programming language, the data transfer would be much faster and I would reach more desirable result.

Now the developed system might be used for the control demonstrations in various courses presented in the Department of Automation Science and Engineering at Tampere University of Technology.

BIBLIOGRAPHY

[1] Wang, Fei-Yue and Derong LIU. Networked control systems: theory and applications. London: Springer, 2008, xviii, 344 p. ISBN 9781848002159.

[2] Quancer Inc. *www.quanser.com/* [online]. 2014 [cited 18 Nov 2015]. Available from URL: <http://www.quanser.com/Content/CoursewareNavigators/qubeservo_matlab/Courseware/Setup/QUBE-Servo%20User%20Manual.pdf>.

[3] Quancer Inc. *www.quanser.com/* [online]. 2016 [cited 18 Nov 2015]. Available from URL: <http://www.quanser.com/history>.

[4] Quancer Inc. *www.quanser.com/* [online]. 2016 [cited 27 Feb 2016]. Available from URL: <http://www.quanser.com/products/q1-crio>.

[5] National Instruments. *www.ni.com/* [online]. Oct 2015 [cited 27 Feb 2016]. Available from URL: <http://www.ni.com/pdf/manuals/375233f.pdf>.

[6] National Instruments. *www.ni.com/* [online]. Oct 2015 [cited 27 Feb 2016]. Available from URL: <ftp://ftp.ni.com/pub/branches/us/quanser/quanser_q1crio_module_quick_start_guide_labview.pdf>.

[7] Kurose, James F. and Ross, Keith W. *Computer networking: a top-down approach featuring the Internet.* Boston: Addison-Wesley, c2001, xxiv, 712 s. ISBN 0201477114.

[8] The MathWorks, Inc. *www.mathworks.com* [online]. Published 10 Jul 2013. Updated 07 Aug 2013 [cited 23 Dec 2015]. Available from URL: <http://www.mathworks.com/matlabcentral/fileexchange/42567-matlab-and-labview-data-exchange-over-tcp-ip>.

[9] The MathWorks, Inc. *www.mathworks.com* [online]. Published 27 Oct 2008. Updated 22 Jul 2010 [cited 23 Dec 2015]. Available from URL: <http://www.mathworks.com/matlabcentral/fileexchange/21908-simulink%C2%AE-real-time-execution>.

[10] National Instruments Corporation. *www.zone.ni.com* [online]. June 2010 [cited 16 Dec 2015]. Available from URL: <http://zone.ni.com/reference/en-XX/help/371361G-01/glang/flatten_to_string/>.

[11] Quancer Inc. *http://www.quarcservice.com/* [online]. [cited 27 Feb 2016]. Available from URL: <http://www.quarcservice.com/ReleaseNotes/files/>.

[12] Franklin, Gene F., J. David Powell and Michael Workman. Digital control of dynamic systems. page 12, 110-114,280-282. Reading, Mass.: Addison-Wesley Pub. Co., c1980. ISBN 0201028913.

[13] *http://www.uta.edu* [online]. Updated: Monday, Dec 24,2012 [cited 24 Apr 2016]. Available from URL: <http://www.uta.edu/utari/acs/ee5321/2013%20notes/2%20lqr%20dt%20and%20sampling.pdf>.

[14] The MathWorks, Inc. *http://www.se.mathworks.com/* [online]. [cited 5 Apr 2016]. Available from URL: <http://se.mathworks.com/help/control/ref/dlqr.html>.

[15] Quancer Inc. *https://faculty.nipissingu.ca/* [online]. 2013 [cited 6 Apr 2016]. Available from URL: <https://faculty.nipissingu.ca/bjs/2107/_LabView/_QUBE%20projects/QUBE-Servo%20Workbook%20(Student).pdf>.

[16] National Instruments Corporation. *www.zone.ni.com* [online]. August 2014 [cited 6 Apr 2016]. 374160B-01. Available from URL: <http://zone.ni.com/reference/en-XX/help/374160B-01/vsmithelp/mit_vsmithelp_boilerplate/>.

[17] The MathWorks, Inc. *www.mathworks.com* [online]. [cited 11 Apr 2016]. Available from URL: <http://se.mathworks.com/products/dsp-system/>.

[18] The MathWorks, Inc. *www.mathworks.com* [online]. [cited 11 Apr 2016]. Available from URL: <http://se.mathworks.com/products/instrument/>.

LIST OF SYMBOLS AND ABBREVIATIONS

AI Analog Input.
AO Analog Output.

CST Control System Toolbox.

DAQ Data Acquisition.
DC Direct Current.
DIN Deutsches Institut für Normung.
DLL Dynamic Link Library.

EI Encoder Input.

I/O Input/Output.
IP Internet Protocol.

LQR Linear Quadratic Regulator.

MIMO Multiple Inputs Multiple Outputs.
MIT Model Interface Toolkit.

NCSs Network Control Systems.
NI National Instruments.

OS Operating System.

PWM Pulse Width Modulation.

QCRP Quanser Rapid Prototyping.

SISO Single Input Single Output.
SVFB State-Variable Feedback.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.
USB Universal Serial Bus.

ZOH Zero Order Hold.

LIST OF APPENDICES

A APPENDIX - MAIN_MINIMAL.M

```
-------------------------------------------------------
%connection settings
port=2057; % Server port
host='localhost';
samplingrate=10; %Samplingrate in ms

%adding library "Soft Real Time" to Simulink
slblocks

%loading Simulink Model and starting simulation in RT
load_system('rtc_example'); %loading model
data_received=0;
%setting initial value
set_param('rtc_example/Constant','Value','data_received');
set_param('rtc_example','SimulationCommand','Start');

while(true)
    %getting parameter from Simulink to Workspace
    rto = get_param('rtc_example/To Workspace', 'RuntimeObject');
    data_to_send = rto.InputPort(1).Data;
    data_received=exchangeData(port,host,samplingrate,data_to_send)
    if(numel(data_received)>0)
        %Setting parameter in Simulink
        set_param('rtc_example/Constant','Value','data_received');
    end;
end
```

B APPENDIX - EXCHANGEDATA.M

```
------------------------------------------------------------
function [data_received]=exchangeData(port,host,samplingrate,data_to_send)
%exchangeData - This code allows to exchange data bewteen Matlab
%and Labview over TCP-IP
%
% Inputs:
%    output_port - port for communication
%    host - host description (only tested with 'localhost'
%    samplingrate - samplingrate for data exchange
%    data_to_send - data you want to send to Labview
%
% Output:
%    data_received - Data from MATLAB

%------------- BEGIN CODE --------------
tic
% java import
import java.io.*
import java.net.Socket

ClientSocket = [];
message=data_to_send;

while true

    try

        %Client socket - it initiates the TCP connection between client
        %and server
        ClientSocket = Socket(host, port);

        %Creating an output stream
        output_stream   = ClientSocket.getOutputStream;
        d_output_stream = DataOutputStream(output_stream);

        %Sending packets
        d_output_stream.writeDouble(message);
```

```
        d_output_stream.flush;

        % Creating an input stream to receive response from the Server
        input_stream   = ClientSocket.getInputStream;
        d_input_stream = DataInputStream(input_stream);

        %Reading data and saving to variable data_received
        data_received=d_input_stream.readDouble;

        %Closing socket
        ClientSocket.close;
        break;

    catch
        if ~isempty(input_socket)
            input_socket.close;
            fprintf(1, 'NO CONNECTION\n');
        end
    end
end

time_needed=toc;
fprintf(1, '\n Time needed: %d\n',time_needed);

if time_needed<samplingrate/1000
    pause(samplingrate/1000-time_needed);
end

end
```

C APPENDIX - QUBE-SERVO.VI