NTNU
Det skapende universitet

Optimering og design av kontroll-software for Schunk PG-070

Anders Ringstad

Master i teknisk kybernetikk
Innlevert: juni 2014
Hovedveileder: Anton Shiriaev, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

0.1 Problem Description

Project title: Optimization of the control system software for the multifunctional robotic device Schunk PG-070.

Trivial friction forces caused by gears, grit and material dynamics are a problem for robotic mechanisms.
An adaptive controller is to be implemented and tested on the Schunk PG-70 gripper, to compensate for this friction and help with accurate positioning control.

Assignment given: 6. January 2014
Supervisor: Anton Shiriaev, ITK

0.2 Preface

This thesis presents the work done during the spring of 2014 as a part of my Master of Science degree at the department of Engineering Cybernetics at the Norwegian University of Science and Technology (NTNU). It is a continuation of my project in TTK4550, and focused on the Schunk Pg70 robotic gripper.

I would like to thank my supervisor, Anton Shiriaev, his assistance has been invaluable and his patience astounding.

I would also like to thank Anton Pyrkin for many hours of assistance at the lab both during the TTK4550 project and continuing through my thesis, as well as Stepan Pchelkin for assistance and patience at the lab.
Thank you.

Anders Ringstad
Trondheim, June 30, 2014

0.3 Summary

The objective of this thesis was to implement an adaptive controller developed by Anton Shiriaev and Anton Pyrkin, to compensate for the friction forces found internally in the Schunk PG-70 robotic gripper. The goal was to implement the adaptive controller as part of a position control system.
A brief introduction to the gripper, the IRB140 robot and the robot programming suite, as well as mentions of some of the more important stability analysis methods used in proving the controller effectiveness.
The next part describes the implementation in RAPID code, followed by the results when the program was tested on the gripper. Due to the gripper malfunctioning, further tests had to be done via simulation.
The results show the potential limitations of real-time control via RobotStudio and RAPID, while showing that the adaptive controller does have the desired effect on the simulated gripper.

Contents

List of Figures

Chapter 1

Preliminaries

Some relevant topics concerning the rest of this report will be introduced here. These topics include information about the PG-70 gripper along with its hardware and software requirements, information about the ABB IRB140 robot, as well as the IRC5 robot controller and RobotStudio. A few words will also concern the topic , of stability of dynamic systems. The introductions will be necessarily brief, and the reader is advised to read *Robust Adaptive Control* [2] by Ioannou and Sun, *Robot Modeling and Control* [4] by Vidyasagar, Spong and Hutchinson, *Modeling and Simulation for Automatic Control* [3] by Egeland and Gravdahl, as well as [6], [9] and [7] for more in-depth information on these topics.

1.1 Schunk PG-70 Gripper

The Schunk PG-70 robotic gripper is a servo-electric, 2-finger parallel gripper intended to grip and to reliably hold objects. Both fingers are connected to one brushless DC-motor through a spindle, allowing for a conversion of the rotational motion of the motor to the linear motion of the fingers. The single motor does not allow for independent control of each finger. Control circuitry along with an encoder and a communications interface, allow for direct control of the gripper finger position.
The module also supports direct control of the applied current, allowing for custom controllers to be developed for position and/or force control. The communications interface allows the module to send and receive data over RS-232, CAN-Bus and PROFIBUS-DP. PROFIBUS-DP is used for communication throughout this project.
Communication between the module and its controller is dictated by a protocol set down by the manufacturer, the Schunk Motion protocol. This protocol dictates the data formats and message structures required for communication with the gripper.

1.1.1 Data Format

Data is represented in floating point values according to the IEEE "Standard for Binary Floating-Point Arithmetic" (IEEE 754). Each number is represented by a 32 bit value consisting of three parts;

Mantissa (23 bits)	Exponent (bit 8 bits)	Sign (1 bit)
f	e	s

and are calculated by the formula;

$$(-1)^s * 2^{e-127} * (1.f)_{bin}$$

All data sent from the gripper will also be of the "little Endian" format, meaning the least significant byte of the data is stored in the lowest memory address allocated in the memory.

Figure 1.1: The Schunk PG-70 gripper

Negative numbers are represented by the two's complement method with a leading 1 (sign bit), positive numbers with a leading 0. Data sent to the gripper must also conform to these properties to be interpreted correctly.

1.1.2 Data Frames

Data frames of the Schunk Motion protocol consist of a D-Len segment (1 byte) indicating the length of the data frame, a command code byte specific to the command issued or responded to, as well as a parameter segment containing required or optional parameters specific to the command byte. Commands are acknowledged by the gripper with a response message upon being received.

Response messages are usually sent as acknowledgements to commands from the controller or as periodic updates if requested, but impulse messages are generated and sent in the case of errors or the module not being able to complete a command. Using PROFIBUS-DP, some limitations are placed upon the structure and size of the data frames. The master/controller can only send data packets of 8 bytes or less, while the gripper can send packets of 16 bytes or less. Two bytes of response messages sent are also reserved for module state information and a message counter to facilitate the integrity of the communication. The protocol supports fragmentation of messages too large to fit the previously mentioned limitations. For more in-depth information on the Schunk Motion protocol the reader is directed to [6].

1.1.3 Command/Response Example

As an example of sending a command and receiving a response, consider the MOVE POS (0xB0) command.

-	D-Len	Cmd	Parameters	Description
M $\rightarrow S$	0x05	0x0B	0x00 0x00 0x20 0x41	Move to position 10.0
S $\leftarrow M$	0x05	0xB0	0xCD 0xCC 0x04 0x41	Position reached in 8.3 sec.

This example illustrates the use of the D-Len segment, the command codes, as well as how parameters are represented when sent to the gripper.

1.2 IRB 140 Industrial Robot and IRC5 robot controller

The gripper is mounted as the end effector on an IRB 140 Industrial Robot from ABB. The robot has six axes, and can handle payloads up to $6kg$. Controlling the robot is a IRC5 robot controller cabinet, also from ABB. The IRC5 contains all the control (main computer, I/O boards and memory) and power electronics needed by the robot. IRC5 controllers come paired with a remote controller called a FlexPendant. This is a handheld device designed to operate the controller without having a direct network connection to the controller through a desktop computer. It also allows for easily testing programs and controller code on the gripper attached to the robot.

1.2.1 RAPID Programming Language

RAPID is a high level programming language designed by ABB for use in controller applications for ABB robots.
Programs in RAPID are divided into modules, which can be either loaded or unloaded during runtime. Each program consists of several module files as well as a single file detailing the relationship between the modules.
For detailed information on syntax and semantics of the RAPID language, the reader is directed at the manuals [8] [9] [7]. The manuals contain details and examples not given here, as well as several routines and procedures used for controlling the robot itself.

1.2.2 RobotStudio

RobotStudio is a collection of software from ABB allowing for programming and simulation of both real and virtual ABB robots connected to a PC. This software is the main tool used for the design of the gripper controller as well as the required communications interface. The software allows the user to specify digital and analog I/O signals used to communicate with the gripper. Commands written to the specified output addresses are sent to the gripper, and responses can be received by reading the corresponding input signals. Included in RobotStudio is a software package called ScreenMaker which allows for drag-and-drop design of GUI applications for the FlexPendant as well as binding variables from the GUI application to variables or objects in the RAPID code running on the controller.

1.3 Stability

Lyapunov theory concerns the stability of dynamical systems, and seek to make conclusions about system trajectories without solving the actual differential equations.
A Lyapunov function $V(x)$ can often be seen as a generalized energy function for the system in question.

Lyapunovs direct method [4][p.455] and the Meyer-Kalman-Yakubovich (MKY) Lemma are

both useful tools for analysing the stability properties of the dynamical systems in this report. The Meyer-Kalman-Yakubovich (MKY) Lemma [2][p.129] is a useful lemma to help in the choosing of Lyapunov functions or Lyapunov-like functions for systems with LTI and nonlinear parts.

Chapter 2

Adaptive Controller design for the Schunk PG-70 gripper

Presented below is a quick overview of the model used for the gripper, as well as the controller to be implemented. The controller itself is designed by [1], and all proofs along with further details can be found there. The gripper is modelled as a second-order linear model, representing a mechanical system with dry and viscous friction.

$$\mathrm{m(t)} = \mathrm{u(t)} + \mathrm{F(2.1)}$$

In this model, $y \in \mathbb{R}^1$ is the output, $u \in \mathbb{R}^1$ is the control input, m is the inertial parameter, $F \in \mathbb{R}^1$ is the friction force

$$F = -\sigma \ \mathrm{sign}(\dot{y}(t)) - \mu \dot{y}(t),$$

with σ being the coefficient for dry friction and μ being the coefficient for viscous friction. $\mathrm{sign}(q)$ is a multivalued function;

$$\mathrm{sign}(q) = \begin{cases} -1 & q < 0 \\ 1 & q > 0 \\ \vartheta & q = 0 \end{cases}.$$

where ϑ belongs to the interval $\vartheta \in [\vartheta_-; \vartheta_+]$.

A full state feedback control law is to be designed so that all trajectories of the closed loop system are bounded , and convergence to zero for the output variable and its derivative, $\lim_{t\to\infty} y(t) = y_0$ and $\lim_{t\to\infty} \dot{y}(t) = 0$.
y_0 is here assumed to be equal to zero, however in the implemented controller the variable y will instead represent the positional error between the reference position and the actual gripper position.
$\vartheta_- = -1$ and $\vartheta_+ = 1$ is also assumed.
The control law is chosen as;

$$\begin{aligned} u(t) &= -k_1 v(t) - k_2 y(t) - k_3 \dot{y}(t) + \varphi(c_1 v(t) + c_2 y(t) + c_3 \dot{y}(t)) \\ \dot{v}(t) &= \alpha_1 v(t) + \alpha_2 y(t) + \alpha_3 \dot{y}(t). \end{aligned} \tag{2.2}$$

For parameters $\alpha_1 = 0, \alpha_2 = 1, \alpha_3 = 0, c_1 < 0$ and φ chosen as

$$\varphi(z) = \sigma\varphi_0(z)$$
$$\varphi_0(z) = \begin{cases} -1 & z < 0 \\ 1 & z > 0 \\ 0 & z = 0 \end{cases}.$$

the stationary set Λ of the closed loop system is $v = 0, y = 0$ and $\dot{y} = 0$, which reflect the desired state variables of the closed loop system.

The resulting model of the closed loop system can now be expressed as;

$$\dot{x}(t) = Ax(t) + Bw(t)$$

$x(t)$ being the vector $x^T = [v, y, \dot{y}]$,

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\frac{k_1}{m} & -\frac{k_2}{m} & -\frac{k_3+\mu}{m} \end{bmatrix}, B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$
$$w(t) = \varphi(c_1x_1(t) + c_2x_2(t) + c_3x_3(t)) - \sigma\text{sign}(x_3(t)).$$

The control law (2.2) with parameters as specified earlier as well as in equations (31)-(36) in [1] can be shown to guarantee that the system is asymptotically stable.

Choosing

$$c_1 = -1, c_2 = -\rho_1, c_3 = -\rho_2,$$
$$\text{k}_2 = \rho_1 k_1, k_3 = \rho_2 k_1$$

with $\rho_1, \rho_2 > 0$ being coefficients set by the user.

Since the control law 2.2 is dependent on system parameters to be known a priori, a set of equations are developed to estimate these parameters. This gives the completed adaptive controller to be implemented to the gripper control software;

$$\begin{aligned} u(t) &= \hat{k}z(t) + \hat{\sigma}(t)\varphi_0(z(t)) \\ \dot{\hat{k}}(t) &= \gamma_1 z^2(t) \\ \dot{\hat{\sigma}}(t) &= \gamma_2 z(t)\varphi_0(z(t)) \\ z(t) &= -\rho_2\dot{y}(t) - \rho_1 y(t) - v(t) \end{aligned} \tag{2.3}$$

It is proven in [1] that this controller assures that output variables $y(t)$ and $\dot{y}(t)$ goes asymptotically to zero along with $\hat{k}(t)$, $\hat{\sigma}(t)$, the estimated values of k_1 and σ converging to some constants.

Chapter 3

Adaptive controller implementation

The gripper itself has a nontrivial amount of friction due to the physical interaction of the different parts and mechanisms. This can stem from gears, dirt and grit in the mechanism or from the DC-motors themselves, as well as general static friction between sliding materials. The plots below highlight the problem with these friction forces. Figure 3.1 is a simulation of a Simulink model built on (2.1) combined with a simple PD controller. The resulting plots show how when the gripper position gets close to the reference position, the control input calculated becomes too small to overcome the friction forces present in the gripper, and the gripper stops moving, while the control input remains at a nonzero value, unable to move the gripper fingers.

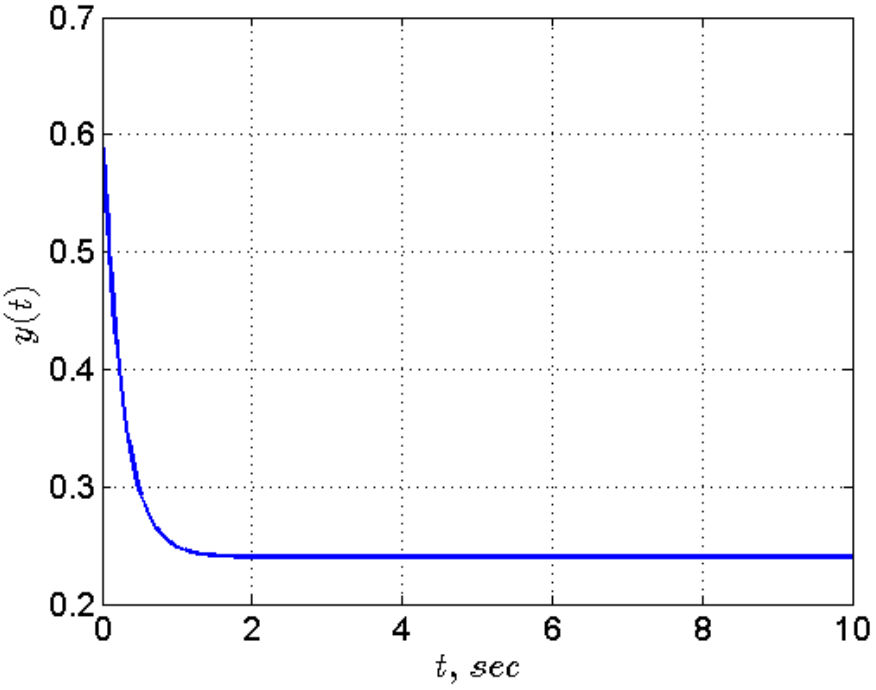


Figure 3.1: Simulated PD-controller, position $y(t)$

The adaptive controller was implemented in RAPID on the IRC5 robot controller connected to the gripper. Following the work of Anton Pyrkin and Anders Ringstad in [5], an interrupt based controller was deemed to be the best solution, as this allows for high priority of the control calculations. Except for setting the position reference input to the controller, the program does all its work during trap routines triggered by periodic interrupt signals. RAPID itself limited the frequency of these interrupts to 10Hz, as to not overflow the interrupt queue and drop mes-

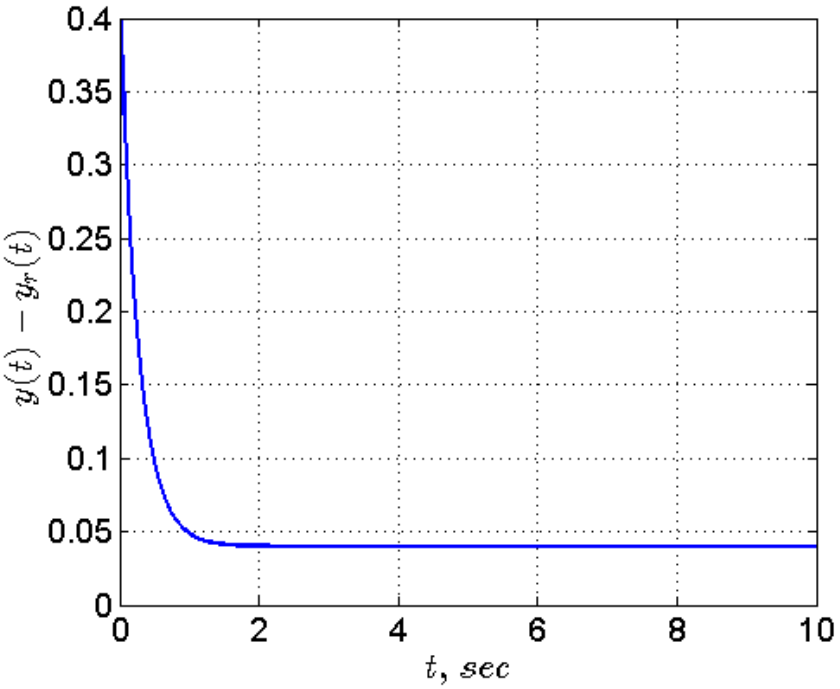
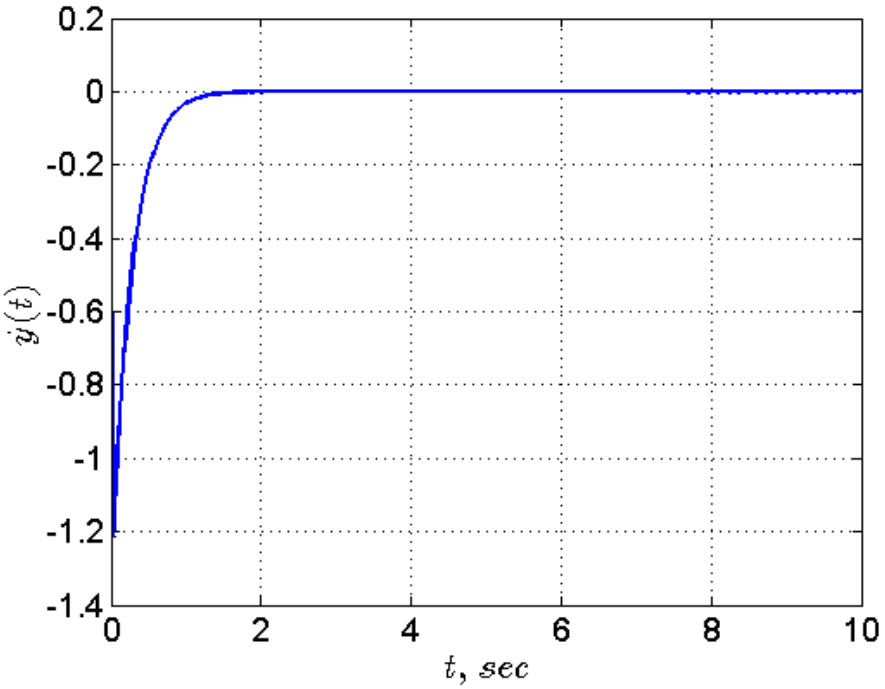


Figure 3.2: Simulated PD-controller, position error $y(t) - y_0$



sages. Depending on the calculations performed during the interrupt, the possible frequency was determined to be somewhere between 5 and 10Hz, as the program would send an error message and stop if overflow of the interrupt queue was detected. Some approximations were also made specifically for the RAPID implementation, as the controller depended on $v(t)$ the integral of the output variable $y(t)$. The integral was approximated by accumulating the approximated definite integral between controller time-steps. A trapezoidal rule approximation was used, as

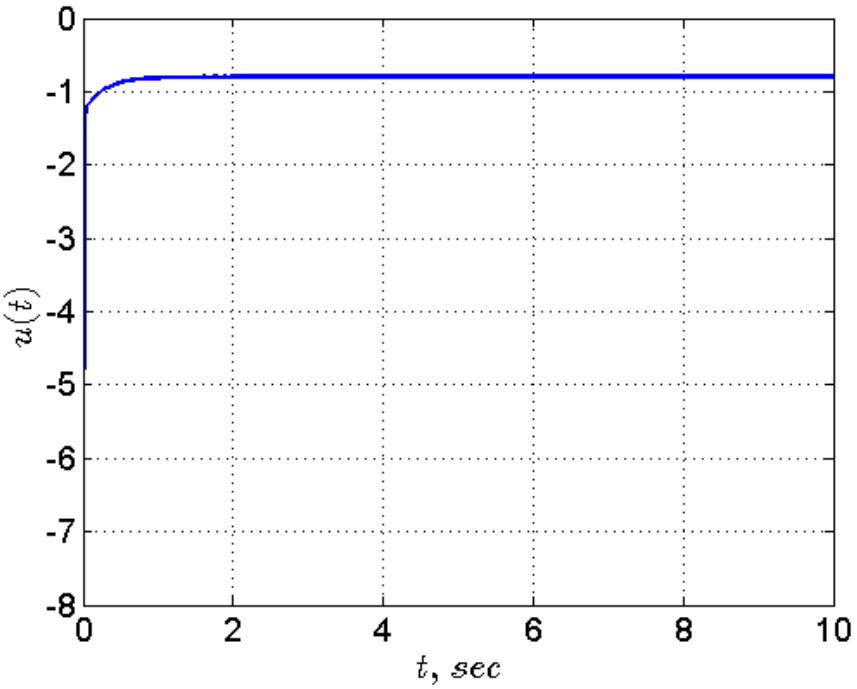


Figure 3.4: Simulated PD-controller, control input $u(t)$

this calculation has low time- and data complexity. Other, more precise approximations were briefly tested, but could not reliably run without dropping interrupt messages.
The main module (MainModule.mod) contains the main procedure, which is run when the program is started. The main procedure connects connects the control and logging trap procedures to timers, allowing them to be called periodically in the background. The gripper then executes a position movement (controlled by the grippers internal software), and a command is sent to the gripper to update its state (position, velocity) with a given frequency. The desired reference position is set and the adaptive controller started. The main procedure then goes into an infinite loop, letting the interrupts handle the control.
When the timer for the controller triggers, an interrupt is sent and the corresponding TRAP procedure is run. The InterruptController.mod contains the procedure for calculating the grippers control input. Each time the TRAP procedure is run, state data sent from the gripper is read and saved to variables in the RAPID program. These values are then used for the required approximations and calculations. It is important to notice that the value being fed into the controller the grippers position, is actually the positioning error relative to the reference position. The controller attempts to drive this error to zero, resulting in a tracking controller.

A rough relation between the applied current and the force measured over the jaw of the gripper was found, $F = 0.01u$ giving the best results. The log_file TRAP procedure contains the necessary code to log signal data to a file, to be imported to MATLAB later.
The GripControl module contains the procedures responsible for sending commands to the gripper, while GripGetState.mod handles the conversion of the data received from the gripper. For a more detailed description of the sending and receiving of commands, the reader is referred to chapter 2 in [5], detailing the functionality of the procedures in GripControl and GripGetState, as well as an explanation of the digital I/O address system used to communicate with the gripper. Figure 3.5 shows the general relationships between each of the modules in the program.

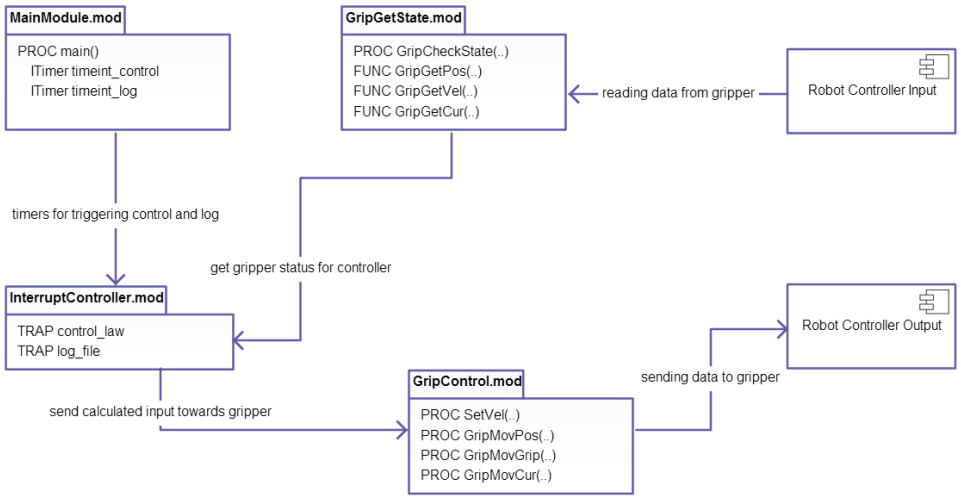


Figure 3.5: How the modules of the program work together

In the interest of easier interaction between the user and controller, a sample GUI was designed in ScreenMaker that allows for basic control of the controller properties through the use of the touch screen on the FlexPendant. The use of this however proved to be of little to no use in the current situation, as switching between the gripper own position movement control and the implemented adaptive controller is not possible without severely modifying the controller program. Figures 3.6, 3.7 show screenshots from the sample GUI run on a virtual FlexPendant in RobotStudio. It was designed by dragging and dropping the various components into the scene, and connecting each of them to a variable in the GUI application, called Application Variables in RobotStudio. The application variables were then connected to corresponding persistent (PERS) variables in the RAPID program, and were set to update the RAPID variables whenever changes occurred in the application variables (i.e. when the user inputs new values). The application file is not included in the report because it does not work with the controller program described in the report. It should however be relatively simple to reproduce a similar GUI application for use with a with a fitting RAPID controller program.

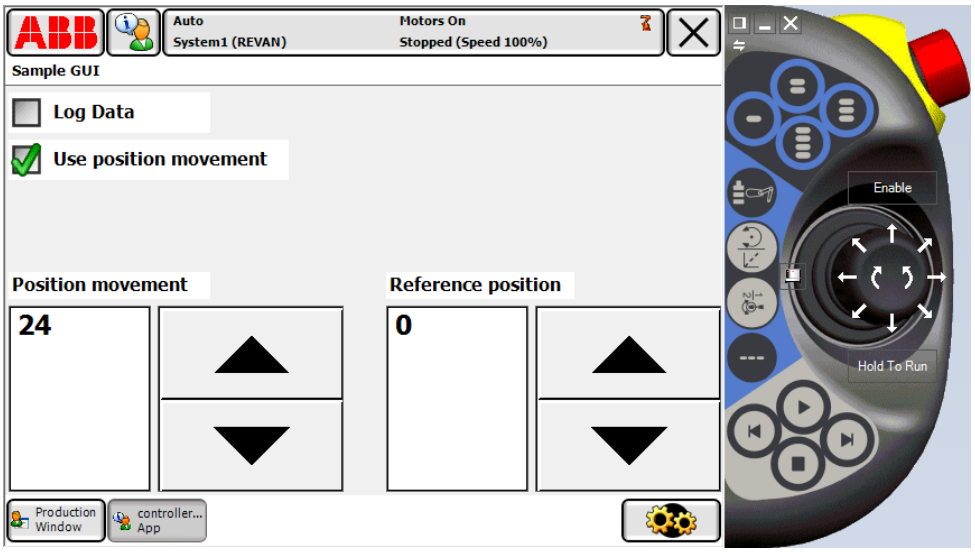


Figure 3.6: Screenshot of the sample GUI application interface

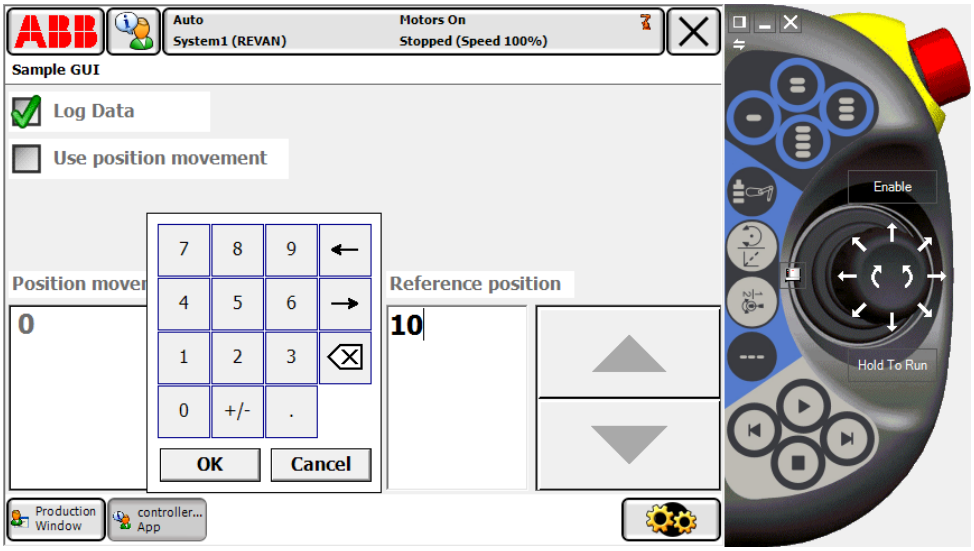


Figure 3.7: Screenshot of the sample GUI application interface

Chapter 4

Experiment Results

The first tests of the controller program showed severe instability, with strong oscillations in the calculated control input resulting in unstable oscillations in the position of the gripper. Figures 4.1-4.6 show the initial results of the controller.

There are several factor potentially contributing to the poor performance of the controller, mainly the frequency updating the control input, the integral approximations and the general structure of the parameter estimations. The estimation of $\hat{k}$ in (2.3) consists of integrating a squared variable, resulting in $\dot{\hat{k}} \geq 0$. Any digital error or noise introduced into this equation will most likely result in aggressive and potentially unbounded growth. This in turn would drive the control output higher as it depends on the calculation of $\hat{k}$ as a controller gain. This could be solved by modifying the adaptive scheme and proving some convergence to zero for the derivative of $\hat{k}$, but a simpler potential solution would be to set upper bounds on the estimated parameters, giving the controller functionality similar to very basic projection. While projection typically is based on knowledge of the plant parameters and their potential bounds are located. It should however also be possible to tune these boundaries as one would a controller parameter, and through this get the desired results from the controller.
This very basic bounding of the estimated parameters was implemented into the controller code, but due to the gripper malfunctioning, further tests concerning the controller could not be made with the physical gripper.

To test the new controller, a Simulink model of the gripper with the modified controller was created. The model was based on the equations (2.3), but was modified with some new constraints to achieve behaviour similar to the gripper. The position and control input was run through a saturation block to limit the amplitude of the signals to what the gripper was able to achieve.

By setting the simulations time-step to a fixed 10Hz, performance similar to the physical gripper was produced. The results (Figures 4.7-4.13) show the same kind of instability and parameter growth, albeit ordered, perhaps due to better integral approximations and less potential for noise and disturbances.
Introducing the parameter bounding to the model resulted in a response fairly similar to the original, but the oscillations are not as strong and more closely grouped around the reference position. The bounding of the parameters removed the unboundedness of the control input, with it now being oscillating within a bounded interval. This is an improvement, however a very small one.

The first significant difference is seen when raising the frequency of the solver to 100Hz. This seems to result in a stable response with the desired behaviour exhibited by the output variables (Figures 4.14-4.20). Both $y(t) - y_0$ and $\dot{y}(t)$ converges to zero (or very close to zero) asymptotically, the speed of the convergence dependent on the controller parameters ρ_1 and ρ_2.

The simulated gripper does not converge exactly to the reference position, but this result is a significant improvement of the ordinary PD-controller results.

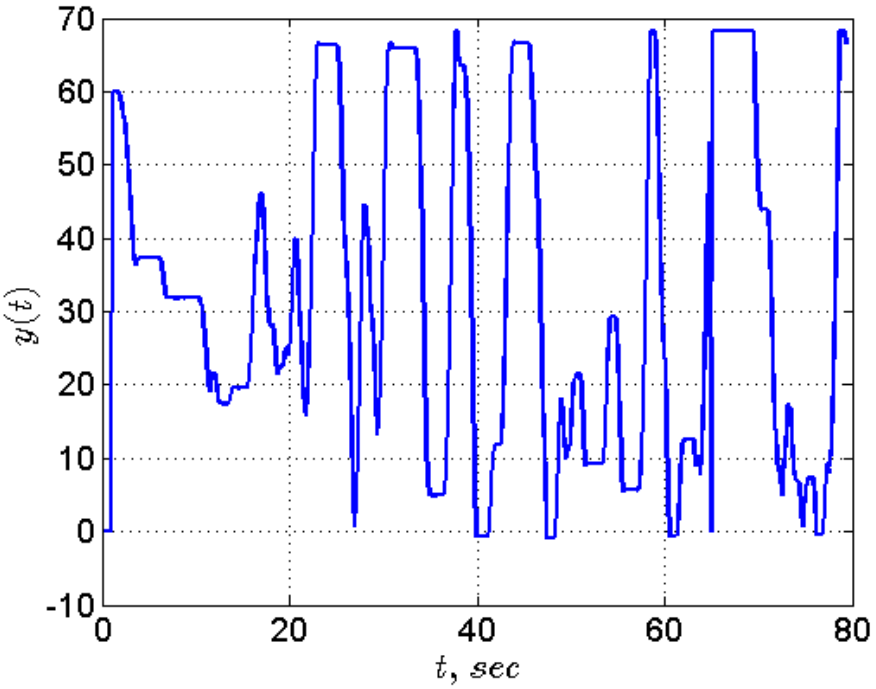


Figure 4.1: Test with real gripper, position $y(t)$

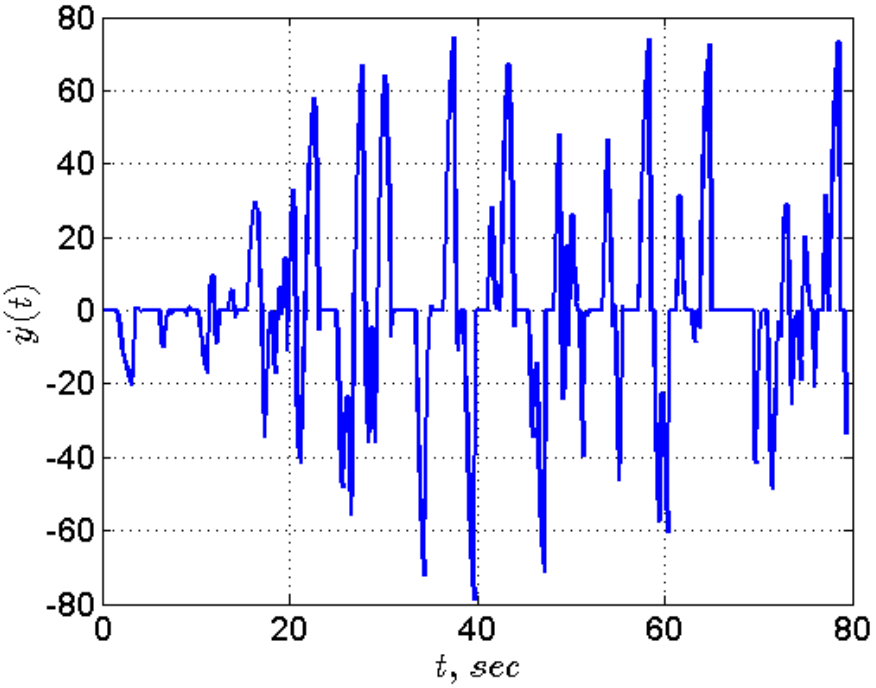


Figure 4.2: Test with real gripper, velocity $\dot{y}(t)$

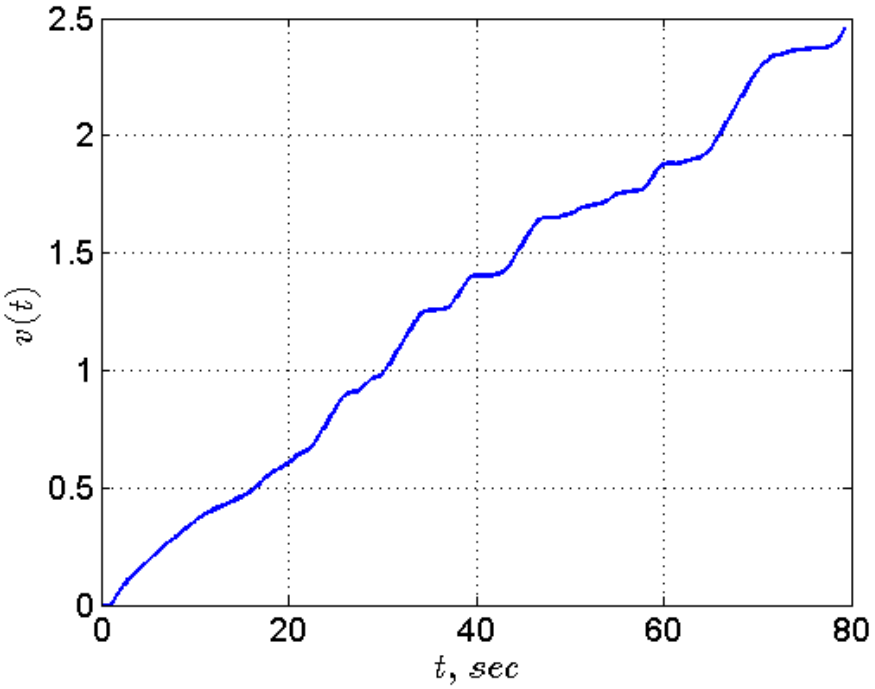


Figure 4.3: Test with real gripper, integral of position $v(t)$

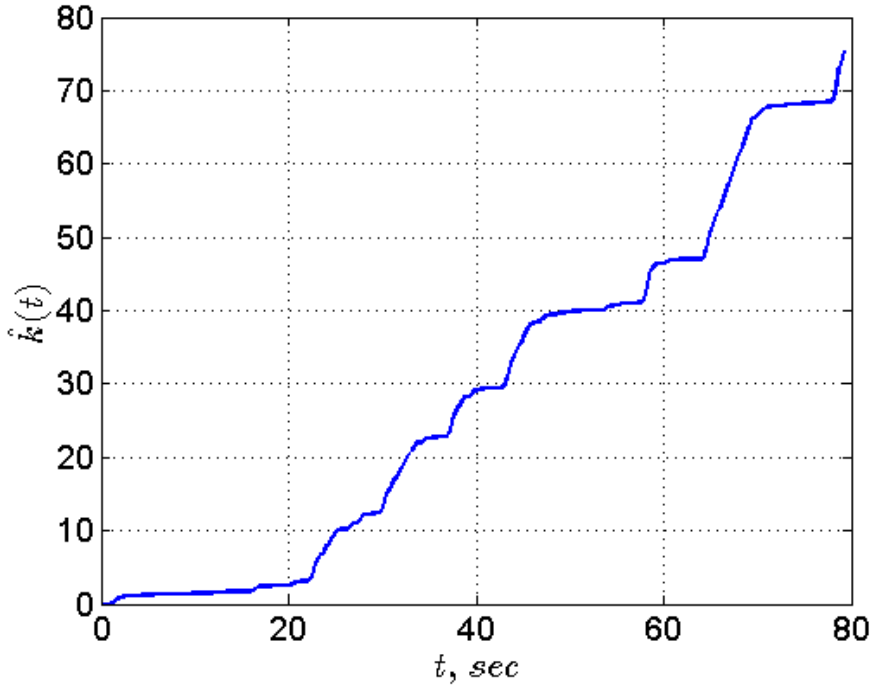


Figure 4.4: Test with real gripper, parameter $\hat{k}(t)$

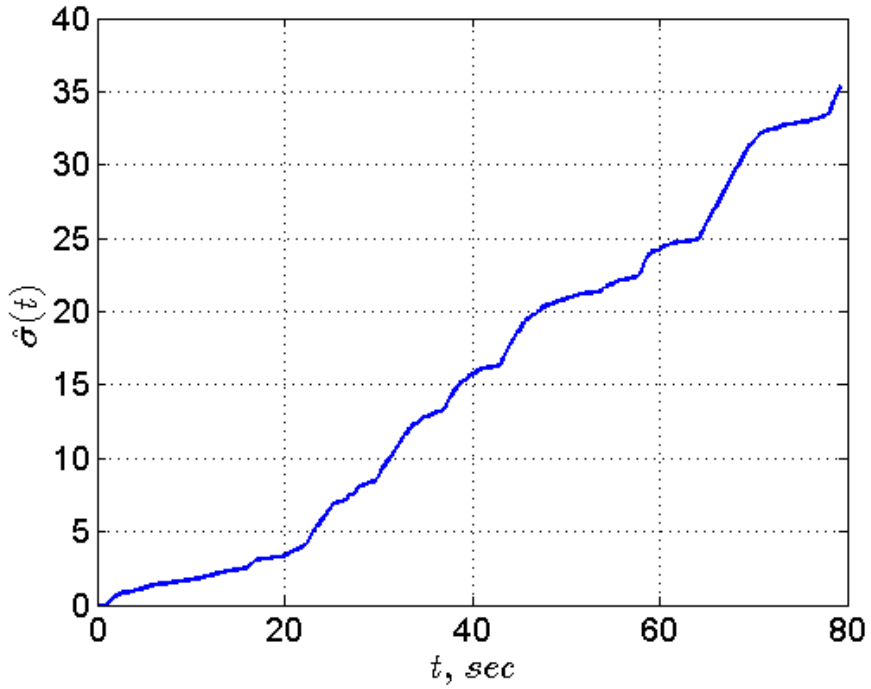


Figure 4.5: Test with real gripper, parameter $\hat{sigma}(t)$

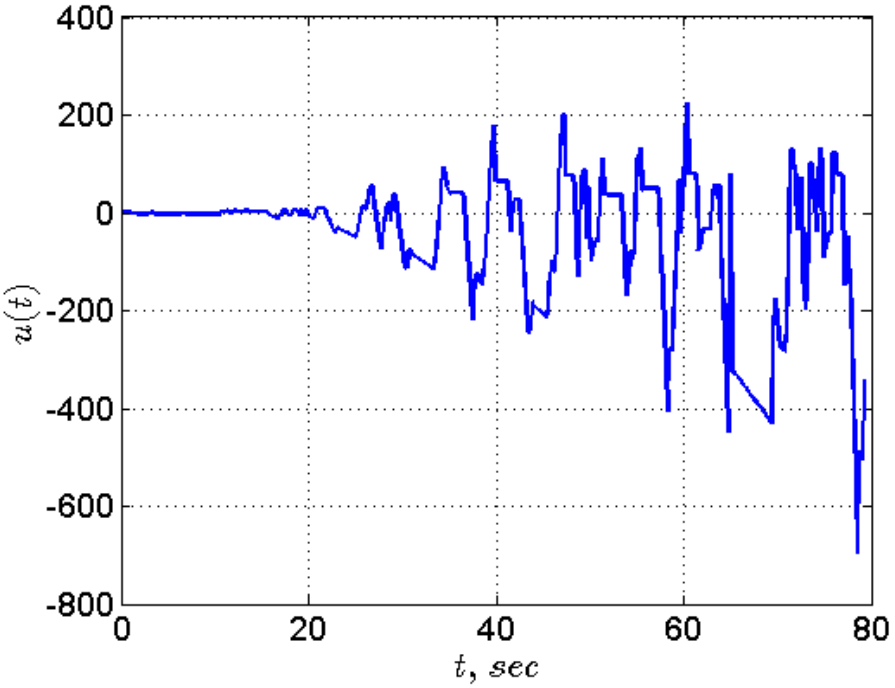


Figure 4.6: Test with real gripper, control input $u(t)$

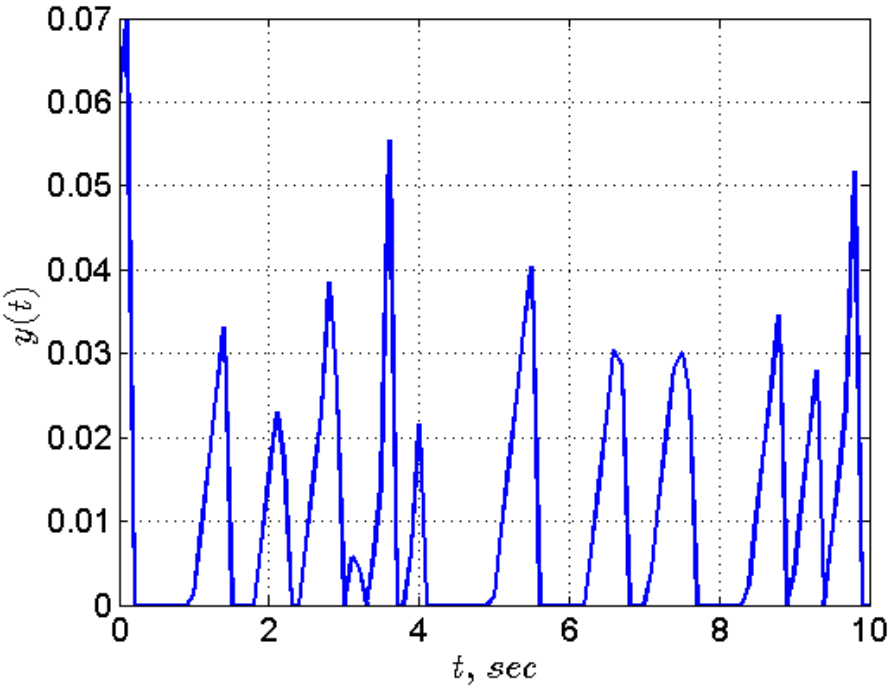


Figure 4.7: Simulation with fixed time step 0.1s, position $y(t)$

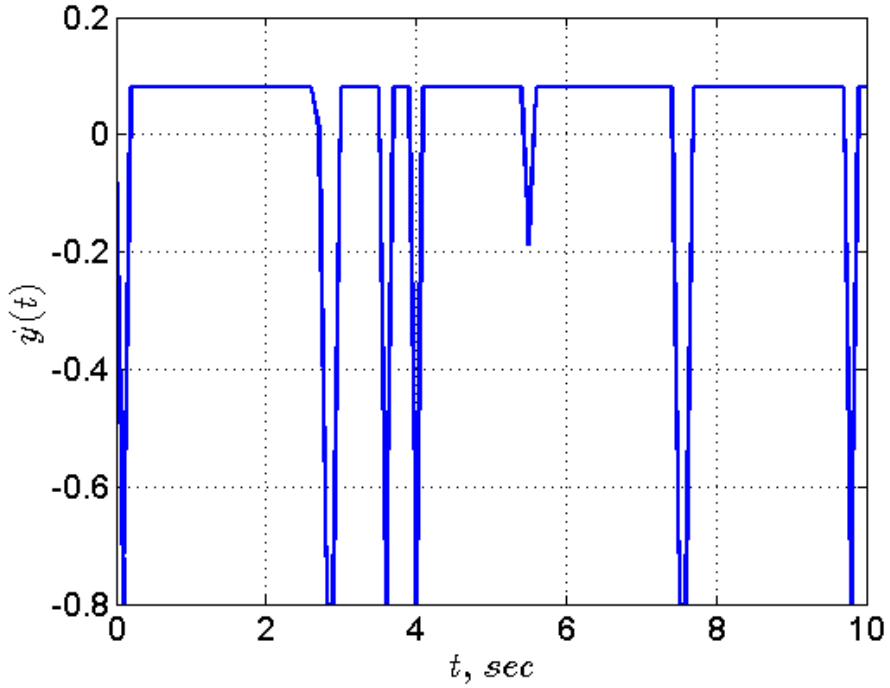


Figure 4.8: Simulation with fixed time step 0.1s, velocity $\dot{y}(t)$

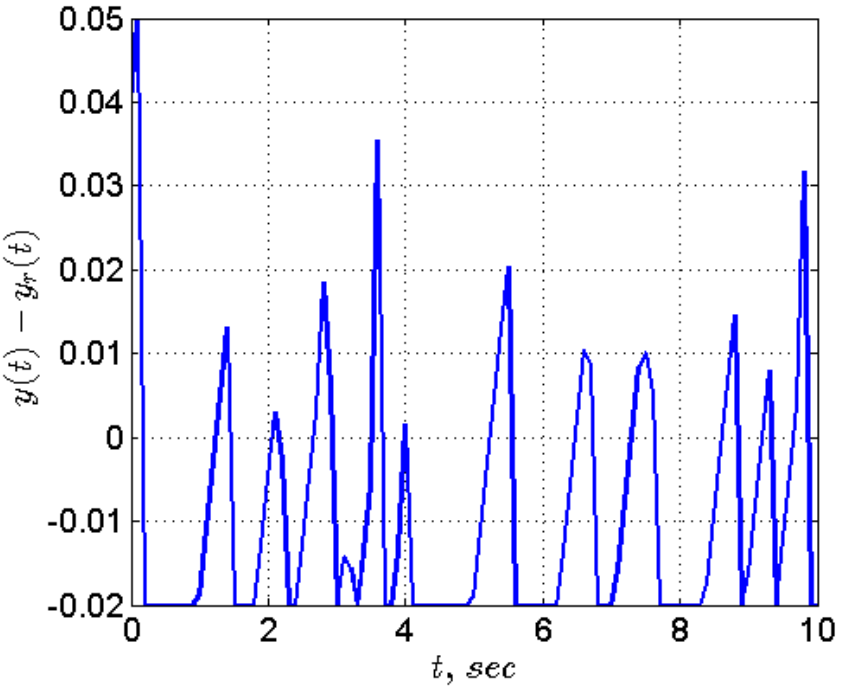


Figure 4.9: Simulation with fixed time step 0.1s, positioning error $y(t) - y_0$

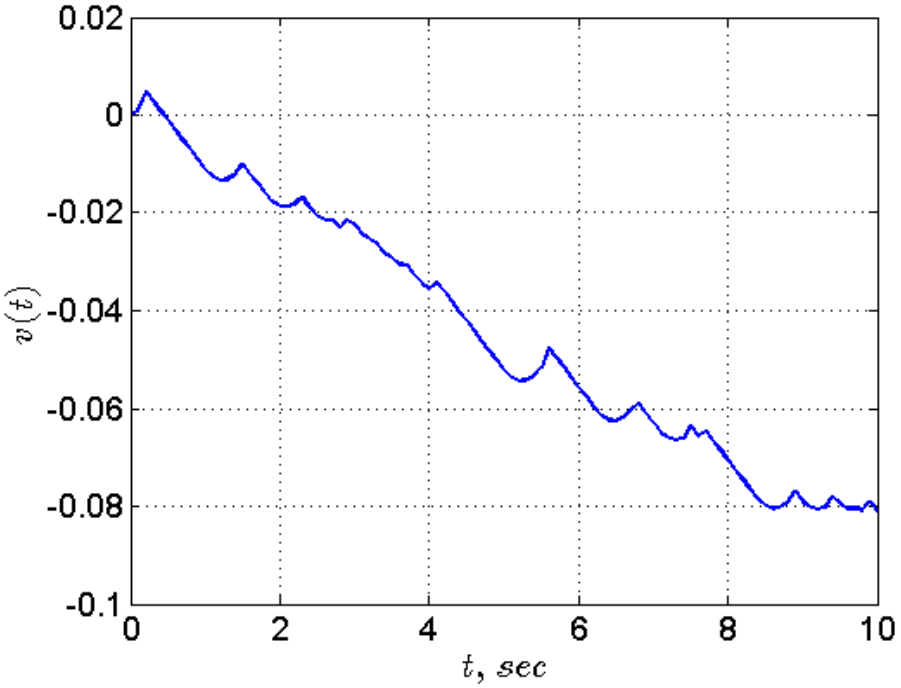


Figure 4.10: Simulation with fixed time step 0.1s, integrated position $v(t)$

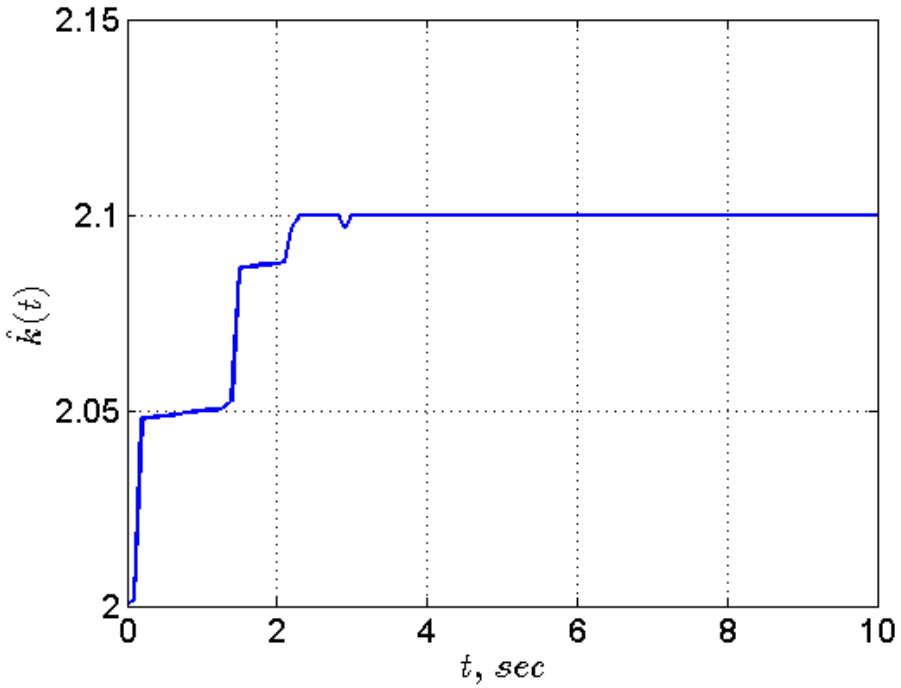


Figure 4.11: Simulation with fixed time step 0.1s, parameter $\hat{k}$

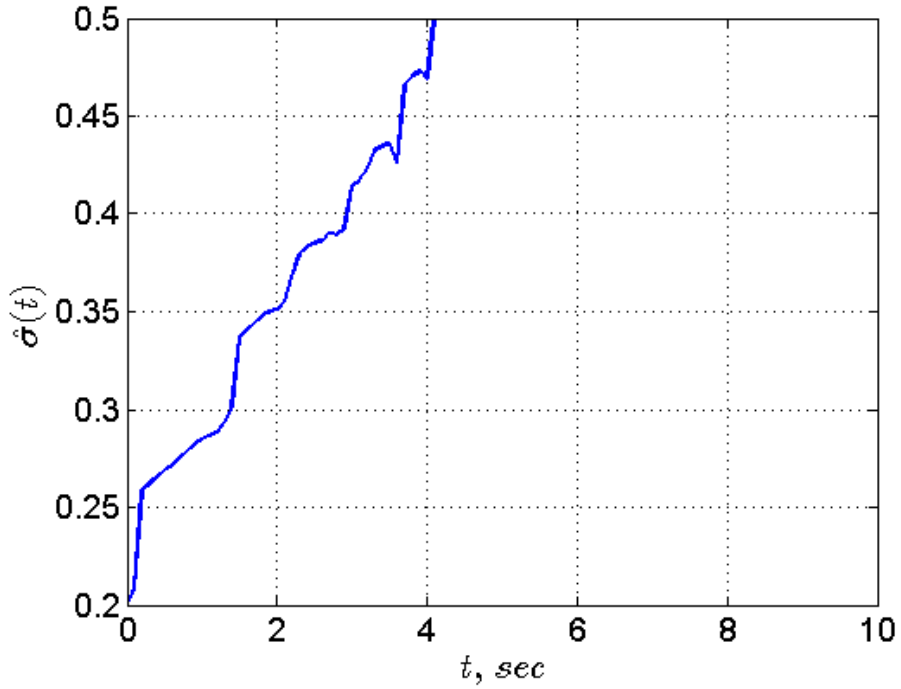


Figure 4.12: Simulation with fixed time step 0.1s, parameter $\hat{sigma}$

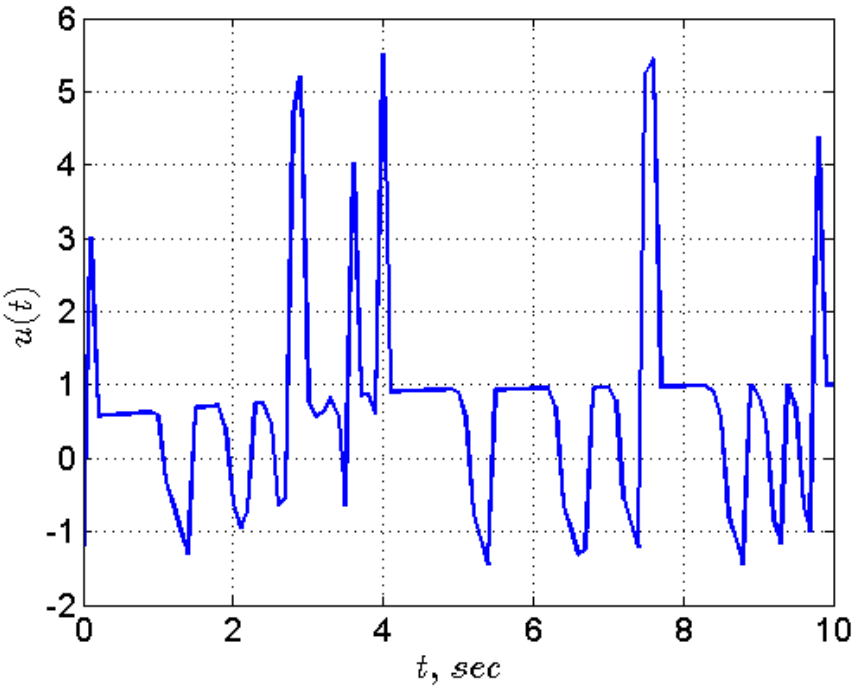


Figure 4.13: Simulation with fixed time step 0.1s, control input $u(t)$

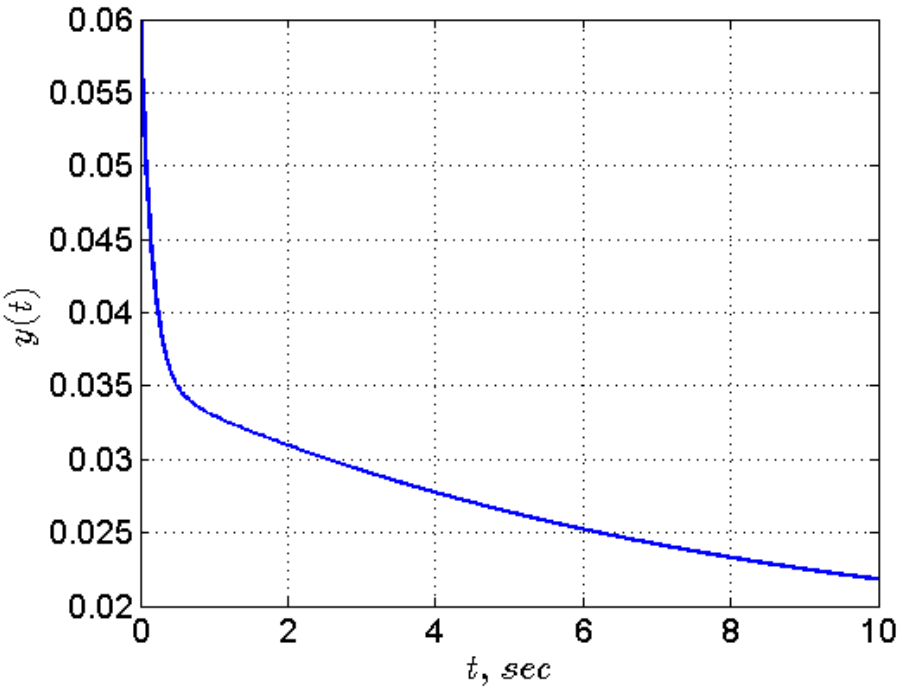


Figure 4.14: Simulation with fixed time step 0.01s, position $y(t)$

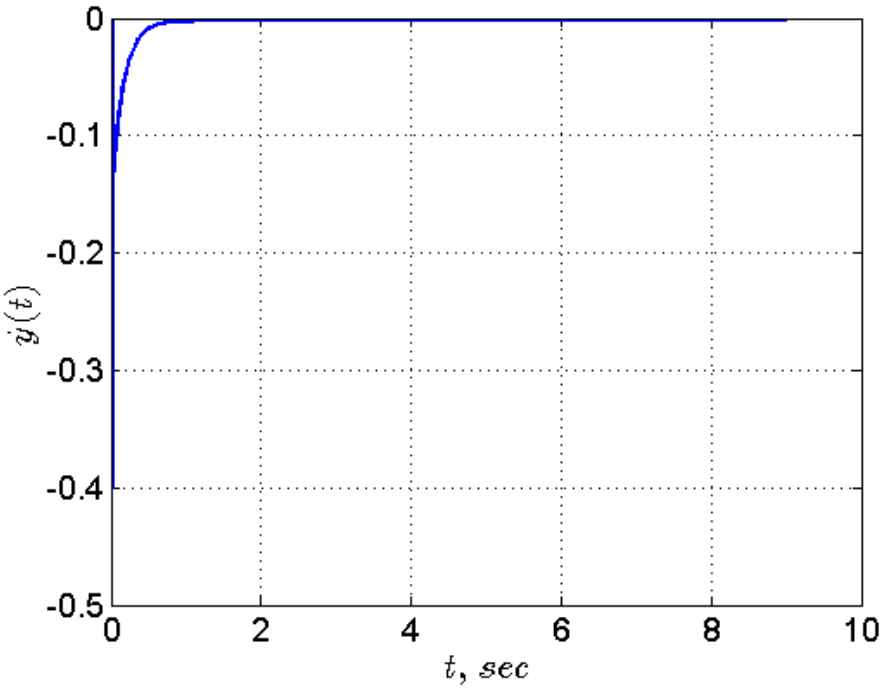


Figure 4.15: Simulation with fixed time step 0.01s, velocity $\dot{y}(t)$

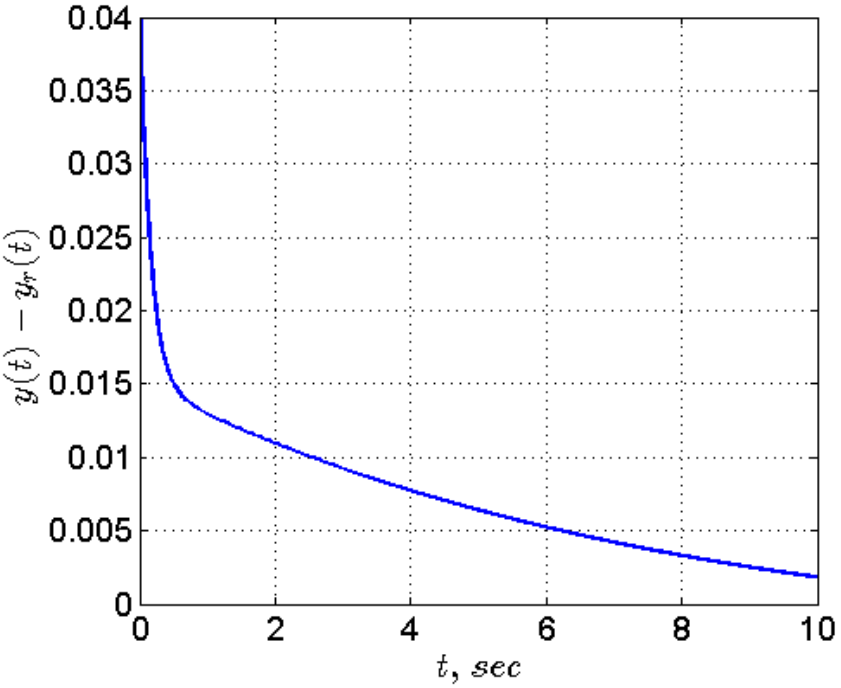


Figure 4.16: Simulation with fixed time step 0.01s, positioning error $y(t) - y_0$

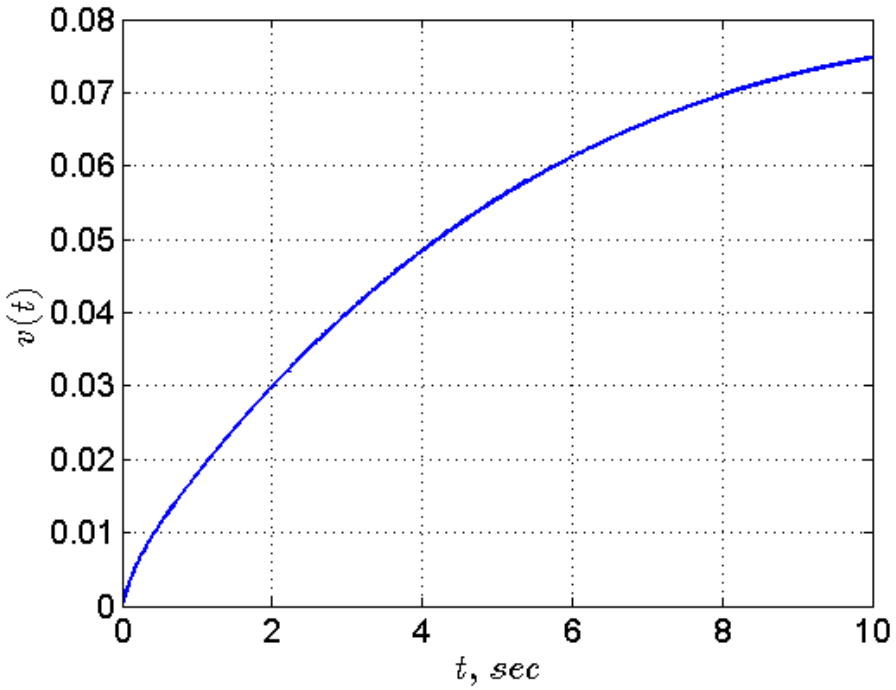


Figure 4.17: Simulation with fixed time step 0.01s, integrated position $v(t)$

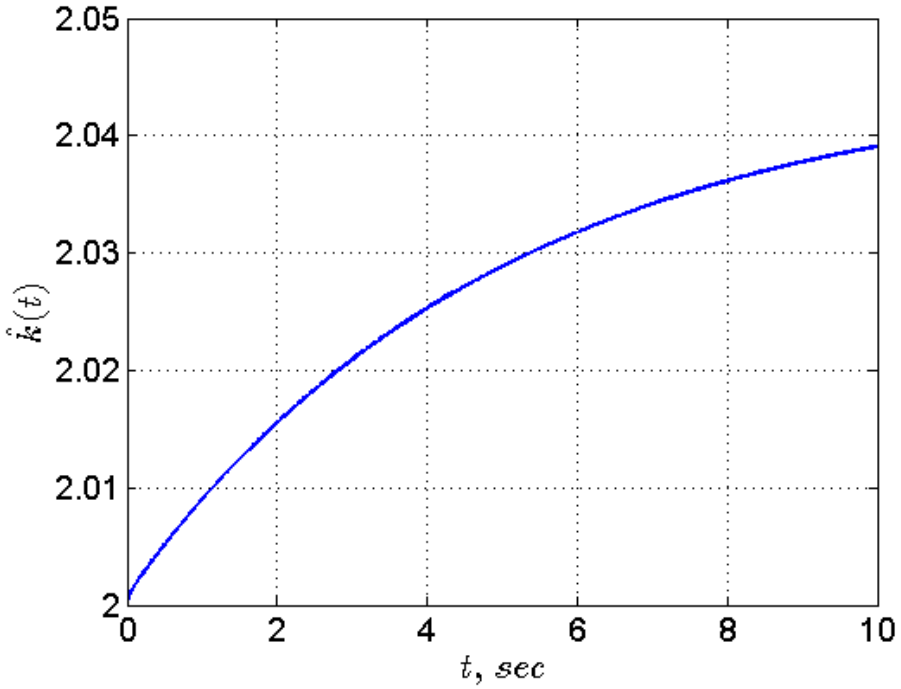


Figure 4.18: Simulation with fixed time step 0.01s, parameter $\hat{k}$

Chapter 5

Conclusions and Further Work

The results clearly show that the implementation of an adaptive controller helped to compensate for the friction in the gripper, and thereby more accurate positioning control was achieved. However, the results also show that the success of the controller is somewhat dependent on the implementation. Software limitations on update timers, rough approximations and the potential of noise and disturbances, make choosing the correct platform or technique for implementation very important for the effectiveness of the solution. The simulations seem to show the effectiveness of the adaptive controller, but as always they are only simulations, and not subject to the hazards of physical testing.
The controller itself showed good results in the simulation, but improvements might be made by attempting to make it less dependent on the integral and derivative of the output variable, as these are often calculated, and with calculation comes potential errors and approximations.
Seeing the apparent limitations of the RAPID control program, means better and easier solutions might be found in other software/hardware suites. It should also be noted that it might be possible somehow to find another method of driving a control loop via RobotStudio and RAPID, that is not subject to the limitations of the interrupt system.

References

[1] Anton A. Pyrkin, Anton S. Shiriaev, *An adaptive controller with new performance analysis for a mechanical system with an interval-valued friction model.*

[2] Petros A. Ioannou, Jing Sun, Robust Adaptive Control. Prentice Hall, Inc, 1996.

[3] Olav Egeland, Jan Tommy Gravdahl, *Modeling and Simulation for Automatic Control.* Marine Cybernetics, 2011.

[4] M. Vidyasagar, Mark W. Spong, Seth Hutchinson, *Robot Modeling and Control.* John Wiley Sons, Inc, 2006.

[5] Anders Ringstad, *Design and implementation of a basic controller for the Schunk PG-70 robotic gripper.* TTK4550 Term Project at NTNU, 2013.

[6] SCHUNK, *SCHUNK Motion.* 2009.

[7] ABB Robotics, *RAPID Instructions, Functions and Data Types, Technical reference manual.* Revision J, 2010.

[8] ABB Robotics, *RAPID Kernel, Technical reference manual.* Revision G, 2009.

[9] ABB Robotics, *RAPID Overview, Technical reference manual.* Revision J, 2010.

Appendix A

Code

A.1 MainModule.mod

```
1  MODULE MainModule
2          var iodev logfile;
3          var clock clock_prog;
4          PERS num pos_ref ;
5      PERS num pos_move;
6          var num start_flag:=0;
7          var num log_start:=0;
8      VAR string fileName;
9
10         PROC main()
11                 var num stop_button := 0;
12
13         fileName := CDate() + "_" + StrMap( CTime(), ":", "-") + "
               _Anders";
14
15
16                 !Manage timer interruption (may not be less than 0.1
                       sec)
17
18         CONNECT timeint_control WITH control_law;
19                 ITimer 0.1, timeint_control;
20
21     CONNECT timeint_log WITH log_file;
22                 ITimer 0.1, timeint_log;
23
24         Open "HOME:" \File:= "../../Logs/" + fileName + ".dat",
               logfile \Write;
25
26         !SetVel 80, 2.0;
27         !GripMovPos 0, 5;
28         !GripMovPos 60, 5;
29
30                 pos_ref := 20;
31
32                 GripCheckState 0.01;
33
```

```
34          ClkStart clock_prog;
35                  log_start := 1;
36                  start_flag := 1;
37
38                  while stop_button = 0 do
39
40          endwhile
41
42                  stop_button := 1;
43
44      ENDPROC
45
46
47
48
49  ENDMODULE
```

A.2 GripControl.mod

```
1   MODULE GripControl
2
3       PROC SetVel(num velocity,num delay)
4           VAR rawbytes raw_data;
5           VAR dnum velocity_coded;
6
7           SetGO GRIP_go_Cmd,0;
8           ClearRawBytes raw_data;
9           PackRawBytes velocity,raw_data,1,\Float4;
10          UnpackRawBytes raw_data,1,velocity_coded\IntX:=DINT;
11          SetGO GRIP_go_DLen,5;
12          SetGO GRIP_go_Data,velocity_coded;
13          SetGO\SDelay:=0,GRIP_go_Cmd,0xa0;
14          WaitTime delay;
15      ENDPROC
16
17          PROC GripMovPos(num position,num delay)
18          VAR rawbytes raw_data;
19          VAR dnum position_coded;
20
21          SetGO GRIP_go_Cmd, 0;
22          ClearRawBytes raw_data;
23          PackRawBytes position, raw_data, 1, \Float4;
24          UnpackRawBytes raw_data, 1, position_coded \IntX:=DINT;
25          SetGO GRIP_go_DLen, 5;
26          SetGO GRIP_go_Data,position_coded;
27          SetGO \SDelay:=0, GRIP_go_Cmd, 0xb0;
28          WaitTime delay;
29      ENDPROC
30
31      PROC GripMovGrip(num current,num delay)
```

```
32          VAR rawbytes raw_data;
33          VAR dnum current_coded;
34
35          SetGO GRIP_go_Cmd,0;
36          SetGO GRIP_go_Cmd,0;
37          ClearRawBytes raw_data;
38          PackRawBytes current,raw_data,1,\Float4;
39          UnpackRawBytes raw_data,1,current_coded\IntX:=UDINT;
40          SetGO GRIP_go_DLen,5;
41          SetGO GRIP_go_Data,current_coded;
42          SetGO\SDelay:=0,GRIP_go_Cmd,0xb7;
43          WaitTime delay;
44
45          SetGO GRIP_go_DLen,1;
46          SetGO GRIP_go_Cmd,0x8b;
47      ENDPROC
48
49      PROC GripMovCur(num current,num delay)
50          VAR rawbytes raw_data;
51          VAR dnum current_coded;
52
53          SetGO GRIP_go_Cmd,0;
54          ClearRawBytes raw_data;
55          PackRawBytes current,raw_data,1,\Float4;
56          UnpackRawBytes raw_data,1,current_coded\IntX:=UDINT;
57          SetGO GRIP_go_DLen,5;
58          SetGO GRIP_go_Data,current_coded;
59          SetGO\SDelay:=0,GRIP_go_Cmd,0xb3;
60          WaitTime delay;
61
62          !SetGO GRIP_go_DLen,1;
63          !SetGO GRIP_go_Cmd,0x8b;
64          !WaitTime delay;
65          sendAck delay;
66      ENDPROC
67
68      PROC sendAck(num delay)
69          SetGO GRIP_go_DLen,1;
70          SetGO GRIP_go_Cmd,0x8b;
71
72          WaitTime delay;
73      ENDPROC
74
75
76
77
78  ENDMODULE
```

A.3 GripGetState.mod

```
1  MODULE GripGetState!(SYSMODULE, VIEWONLY)
2
3  PROC GripCheckState(num interval)
4      VAR rawbytes raw_data;
5      VAR dnum interval_coded;
6      !
7      SetGO GRIP_go_DLen, 1;
8      SetGO GRIP_go_Cmd, 0x8b;
9      WaitTime 1.0;
10
11     SetGO GRIP_go_Cmd,0;
12     ClearRawBytes raw_data;
13     PackRawBytes interval,raw_data,1,\Float4;
14     UnpackRawBytes raw_data,1,interval_coded\IntX:=UDINT;
15     SetGO GRIP_go_DLen,6;
16     SetGO GRIP_go_Data,interval_coded;
17     SetGO GRIP_go_Data5,0x07;
18     SetGO GRIP_go_Cmd,0x95;
19     WaitTime 1.0;
20 ENDPROC
21
22 !Coverts bytes sent from gripper to real value, and returns this
       value
23 FUNC num GripGetPos(dnum position)
24         VAR rawbytes raw_data;
25         VAR num meas_position;
26         ClearRawBytes raw_data;
27         PackRawBytes position, raw_data, 1, \IntX:=UDINT;
28         UnpackRawBytes raw_data, 1, meas_position \Float4;
29         RETURN meas_position;
30 ENDFUNC
31
32 FUNC num GripGetVel(dnum velocity)
33         VAR rawbytes raw_data;
34         VAR num meas_velocity;
35         ClearRawBytes raw_data;
36         PackRawBytes velocity, raw_data, 1, \IntX:=UDINT;
37         UnpackRawBytes raw_data, 1, meas_velocity \Float4;
38         RETURN meas_velocity;
39 ENDFUNC
40
41 FUNC num GripGetCur(dnum current)
42         VAR rawbytes raw_data;
43         VAR num meas_current;
44         ClearRawBytes raw_data;
45         PackRawBytes current, raw_data, 1, \IntX:=UDINT;
46         UnpackRawBytes raw_data, 1, meas_current \Float4;
47         RETURN meas_current;
48 ENDFUNC
49
50 ENDMODULE
```

A.4 InterruptController.mod

```
1  MODULE InterruptController!(SYSMODULE, VIEWONLY)
2
3      VAR intnum timeint_control;
4          VAR intnum timeint_log;
5
6      VAR bool first_time := TRUE;
7
8          VAR num position;
9      VAR num position_int;
10         VAR num velocity;
11
12     VAR  num set_cur;
13
14     VAR num approxFrom;
15     VAR num approxTo;
16     VAR clock approxClock;
17     VAR num approxTime;
18     VAR num lastApproxTime;
19     VAR num lastPosition;
20     VAR num posError;
21     VAR num lastPosError;
22
23     VAR num u;
24     VAR num k_hat;
25     VAR num k_hat_dot;
26     VAR num k_hat_dot_last;
27     VAR num sigma_hat;
28     VAR num sigma_hat_dot;
29     VAR num sigma_hat_dot_last;
30     VAR num z;
31
32     CONST num gamma_1 := 0.1;
33     CONST num gamma_2 := 0.2;
34     CONST num rho_1 := 15;
35     CONST num rho_2 := 4;
36
37
38         VAR num cur_bound := 0.4;
39
40     TRAP control_law
41         IF start_flag =1 THEN
42             IF(first_time) THEN
43                 ClkStart approxClock;
44                 position := GripGetPos(GInputDnum(Grip_gi_Param1));
45                 velocity := GripGetVel(GInputDnum(Grip_gi_Param2));
46                 position_int := 0;
47                 lastApproxTime := 0;
48                 lastPosition := position;
49                 lastPosError := position - pos_ref;
```

```
50
51              k_hat_dot_last := 0;
52              sigma_hat_dot_last := 0;
53
54              first_time := FALSE;
55          ENDIF
56
57          IF GInputDnum(Grip_gi_DLen) = 0x0f AND GInputDnum(
                Grip_gi_Cmd) = 0x95 THEN
58              position := GripGetPos(GInputDnum(Grip_gi_Param1));
59              velocity := GripGetVel(GInputDnum(Grip_gi_Param2));
60          ELSE
61              sendAck 0.01;
62          ENDIF
63
64          approxTime := ClkRead(approxClock);
65          posError := position - pos_ref;
66          position_int := position_int + trapezoidal_approx(
                lastPosError/1000, posError/1000, lastApproxTime ,
                approxTime);
67
68          z := -rho_2*velocity/1000 - rho_1*(posError/1000) -
                position_int;
69
70                      !IF k_hat < 4.5 THEN
71              k_hat_dot := gamma_1*z*z;
72          !ELSE
73          !     k_hat_dot := 0;
74          !ENDIF
75
76
77          !IF sigma_hat < 2.5 THEN
78             sigma_hat_dot := gamma_2*z*calc_phi(z);
79          !ELSE
80          !     sigma_hat_dot := 0;
81          !ENDIF
82
83          k_hat := k_hat + trapezoidal_approx(k_hat_dot_last ,
                k_hat_dot, lastApproxTime, approxTime);
84          sigma_hat := sigma_hat + trapezoidal_approx(
                sigma_hat_dot_last, sigma_hat_dot, lastApproxTime,
                approxTime);
85
86          u := k_hat*z + sigma_hat*calc_phi(z);
87
88          set_cur := 0.01*u;
89
90
91          IF set_cur > cur_bound  then set_cur := cur_bound; endif
92                  IF set_cur < -cur_bound then set_cur := -
                        cur_bound; endif
```

```
93
94              GripMovCur set_cur, 0;
95
96
97              k_hat_dot_last := k_hat_dot;
98              sigma_hat_dot_last := sigma_hat_dot;
99              lastPosition := position;
100             lastApproxTime := approxTime;
101             lastPosError := posError;
102
103                 ENDIF
104     ENDTRAP
105
106         TRAP log_file
107                 VAR num time;
108          IF log_start = 1 THEN
109                 time:=ClkRead(clock_prog);
110                 Write logfile, CTime()\NoNewLine;
111                 Write logfile, " "\Num:=time\NoNewLine;
112                 Write logfile, " "\Num:=position \NoNewLine;
113                 Write logfile, " "\Num:=velocity \NoNewLine;
114         Write logfile, " "\Num:=position_int \NoNewLine;
115                 Write logfile, " "\Num:=u \NoNewLine;
116         Write logfile, " "\Num:=k_hat \NoNewLine;
117         Write logfile, " "\Num:=sigma_hat \NoNewLine;
118                 Write logfile, " # ";
119          ENDIF
120
121         ENDTRAP
122
123     FUNC num trapezoidal_approx(num f_a, num f_b, num timeA, num
            timeB)
124
125
126         RETURN ((timeB - timeA)*(f_a + f_b)/2);
127     ENDFUNC
128
129     FUNC num calc_phi(num input)
130         IF(input < 0) THEN
131             RETURN -1;
132         ELSEIF(input > 0) THEN
133             RETURN 1;
134         ELSE
135             RETURN 0;
136         ENDIF
137     ENDFUNC
138
139 ENDMODULE
```

Appendix B

Attached Files

Attached with the thesis is a .zip archive with relevant Simulink models and MATLAB code, as well as the RAPID code for the program and all plots.