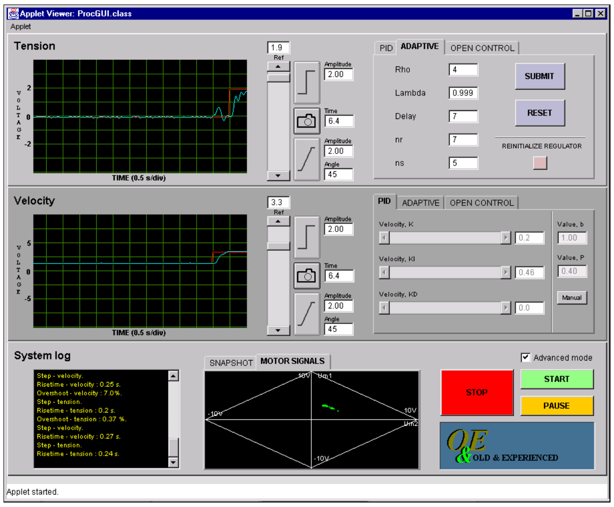
Direct adaptive control of a coupled electric drives apparatus



Authors:	Karim Ahari	F4Sy
	Andreas Lingvall	F4Sy
	Andreas Lundeen	F4E
	Lars-Fredrik Smedberg	F4Sy
	Oskar Wibling	F4Sy

Tutors: Claes Tidestav and Nilo Ericsson

June 6, 1998

Contents

1 Introduction

The goal of the project was to control an experimental process, "Coupled electric drives". Its dynamics is somewhat like when a continuous string of some material (steel, paper etc.) is passed through a molding station. During the course of this the tension and velocity of the material needs to be controlled fairly precisely. In the experimental process a rubber band is attached on three moving wheels in the shape of a triangle. Two of the wheels are powered by direct current motors, while the third is suspended from a spring. This seamingly harmless process proves to be quite a challenge to control. The system is oscillative and the tension of the rubber band is pretty difficult to control. On top of this each of the motor signals affects both the controlled variables (tension and velocity). Our group chose to use direct adaptive control as our main regulating technique.

The control system itself is written as Matlab procedures that run under Linux on a computer that is connected to the coupled electric drives apparatus. The Matlab procedures in turn exchange information with a Java application running on a Windows NT system over a 100 Mbit/s Ethernet. The Java application is meant to be the main interface towards the user and he(or she) should get the complete information about the system without actually seeing it. The computer platform in our case consists of two dual Pentium II 266 MHz computers.

2 Modelling of the process

Since we used the technique of direct adaptive control (and of course PID that everybody had to implement) we did not have to perform any system identification on the process. The regulator parameters are updated on line by making the regulator adjust itself, learning from its mistakes. We just saw the process as something with two input variables and two output variables by using the approximation $U_e = (U_1 - U_2)/2$ and $U_v = (U_1 + U_2)/2$ where U_v=Velocity control signal, U_e=Tension control signal, U_1=Motor one control signal and U_2=Motor two control signal.

3 Regulators

3.1 Theory and design

The lab process is not easily controllable outside the boundaries $\pm 2V$ for the tension control signal and $\pm 5V$ for the velocity control signal because of the great unlinearity there. Since our regulator was only required to work within these limits, however, the work was greatly simplified.

3.1.1 The PID regulator

The PID regulator chosen in this project was extended with two extra parameters **b** and **p**. The **b** introduces an extra zero and affects the overshoot in a positive manner if selected properly. The **p** is the prefilter pole for the input signal and is used to reduce the fast modes.

In order to receive a well trimmed regulator that fulfills the specifications across the whole working range, nine different regulators were trimmed. More exactly the working range was divided into three times three subranges and constants for all of these were stored. The user can in the interface choose between automatic mode, in which the system decides which of the nine presets to use, or manual mode, in which he (or she) sets the constants directly.

3.1.2 The adaptive regulator

The adaptive regulator is of a *CARIMA* type and *Potter's algorithm* was used to get a numerically stable method. To avoid windup problems the program limits the trace of the P-matrix (i.e. $Q^H Q$), by simply turning off the adaption. This is done when the trace of the $Q^H Q$ matrix exceeds an upper bound or when the standard deviation of the estimation error for the last eight samples drops below a certain level.

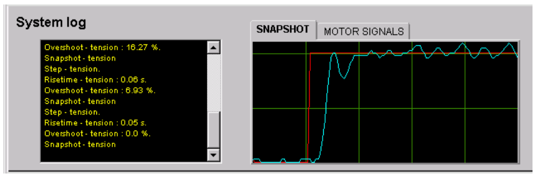
Both of these limits were determined while searching for the optimal regulator (see appendix A fig. (18), (19) and (20)).

3.2 Performance

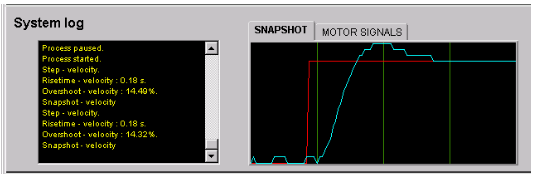
The transfer between different regulators is performed in a bumpless manner. The specifications given are fulfilled for all work points in the range of $\pm 2V$ for the tension control signal and $\pm 5V$ for the velocity control signal. Our advanced regulators (*adaptive control*) are not very fast, but in accordance with the specifications. All the regulators give stable systems with static gain close to 1. For the open loop control it is very difficult to adjust the static gain perfectly but we have done our very best.

3.2.1 The PID regulator

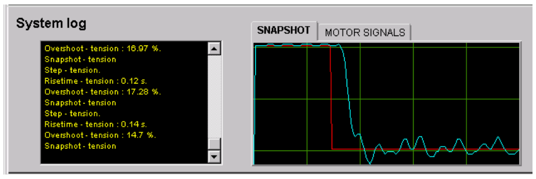
The PID regulators for both tension and velocity are well trimmed i.e. they yield a low rise time and little overshoot. See figure (1) below for an image that shows this.



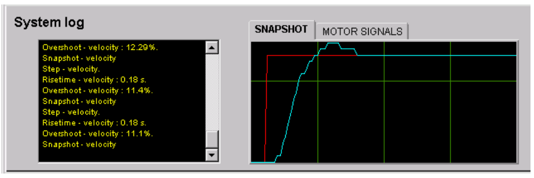
(a) *Step tension PID*



(b) *Step velocity PID*



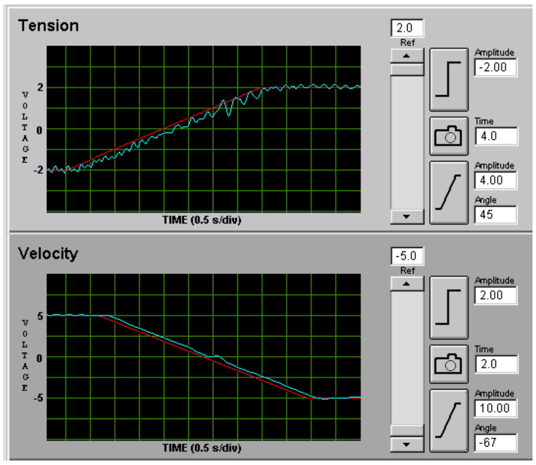
(c) *Step tension adaptive*



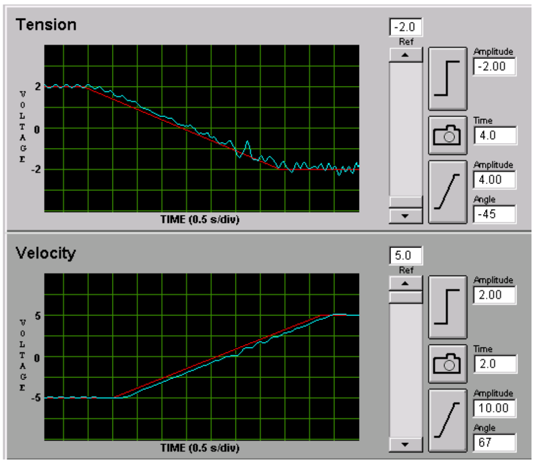
(d) *Step velocity adaptive*

As can be seen in figure (2) the PID regulator also follows the reference signal well when it is a ramp.

Figure 1: *Step responses.*



(a) *Ramp PID*



(b) *Ramp adaptive*

Figure 2: *Ramp responses.*

3.2.2 The direct adaptive regulator

The risetime for the adaptive regulator is below 0.25 seconds for both the velocity and the tension with an overshoot less than 20% as can be seen in figure (1). risetime In figure (2) the ramp response of the adaptive regulator is shown.

3.3 Comparison of control performance

When comparing the direct adaptive regulator with the PID we can conclude that the PID regulates slightly better for the velocity. However for the tension the adaptive regulator is close to perfect while the PID is slightly worse. The reason for the PID being such a good regulator in spite of its simplicity is for the most part the nine different presets as mentioned above.

3.4 Robustness of the developed systems

The adaptive regulator for the tension damps disturbances much better than the open loop and PID regulators. See figure (3) for an example of damping for the adaptive regulator.

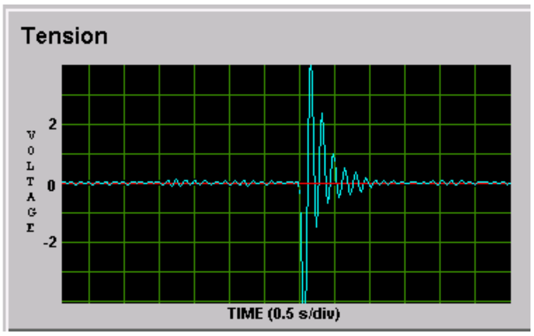


Figure 3: *Disturbance damping adaptive.*

However, the adaptive regulator does not damp as well as one would expect for a regulator of this type. The reason for this is that part of the damping performance has been lost when designing the system to be this fast.

4 The graphical user interface(GUI)

A screenshot of the GUI can be seen in figure (4).

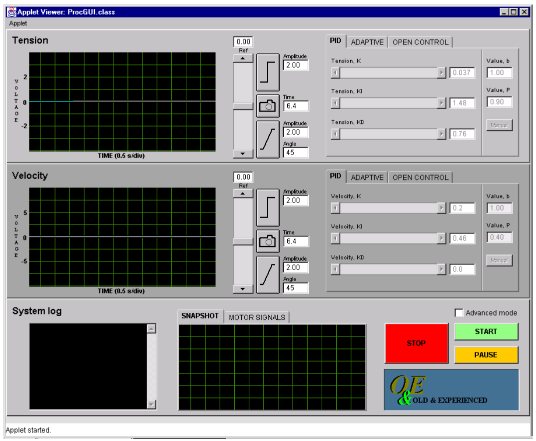


Figure 4: *The graphical user interface(GUI).*

It was initially modelled with the help of *Symantec Visual Cafe* but after a while we started implementing our own algorithms etc. and *Symantec* could no longer modify the code. When we realized this we reduced the role of *Symantec* to being just a very fast Java JIT (Just In Time) compiler. The process did in fact run much faster under Symantec than under for example Netscape.

The GUI has the following main features:

- It presents measured and wanted values of the tension and velocity for the process in two real time diagrams resembling the display of an oscilloscope.
- Everything that plays a significant role in the history of the system is logged in a little terminal window in the GUI. The user can therefore scroll back and see how the system has been running.
- It features the ability to control the reference signals (U_e and U_v) with sliding controls or by typing in a value.
- It has the ability to simulate steps and ramps which can be varied in a very flexible manner.
- The rise time and overshoot are automatically calculated and presented in the terminal window when a step is performed.
- Choice between making only the basic functionality in the GUI user controlled and making all the functionality in the GUI user controlled. This choice is made by marking or unmarking the "Advanced mode" checkbox.
- It gives the user the ability to change regulator mode between open control, PID regulation and adaptive control. Different types of regulators can be used on the tension and velocity respectively and the transfer between modes is fairly bumpless. Parameters for the PID regulator (K, K_i, K_d, P and b) and the adaptive regulator (rho, lambda, delay and polynomial orders) can be set freely by the user. He (or she) can also let the system set the parameters by not choosing advanced mode in the lower control panel. Another option is to choose advanced mode and then to select individually if the system controls the tension, the velocity or both.

- It presents the control signals (U_1 and U_2) in a separate real time diagram. This diagram plots U_1 and U_2 against each other with an afterlighting of 25 old values as can be seen in figure (11).

- A snapshot feature that takes a user defined time of samples from one of the two diagrams (tension or velocity) and projects them to another window in the lower part of the interface. The purpose of this function is to make it possible to study the process in detail at certain points in time without having to halt the updating of some diagram. Therefore the section that is taken as snapshot is scaled to take up as much of the window as possible as can be seen in figure (9).

- It alarms and/or stops the process when the system is oscillative, the rubber band is broken or the spring is broken. The alarm systems are fairly advanced with different levels of warning etc. In the oscillation alarm we start by printing a warning message after about ten large oscillations. If the user does not respond to the warning in time then the control is switched to open loop mode (which is guaranteed to be nonoscillative). When the oscillations have been below a certain magnitude for a period of time the oscillation counter is reset to zero. The alarm systems for the rubber band and spring also have two levels. They first warn the user and switch to open loop control when the values read from the process are abnormally high or low. After they have been above a certain magnitude for a number of samples the process is paused automatically. The alarm systems, in addition to printing out messages in the system log, also show an animated gif with a blinking warning sign and play sampled sounds. All this to alert the user when something is wrong.

- The user can stop the process completely so that both the Matlab side has to be restarted and thereafter the Java side has to be restarted. The user can also pause the process whereby the control signals are reset to zero until the process is started again by the user.

The GUI is designed and coded with the main purpose of being very easy to read and make adjustments to.

5 Structure of the programs

Since the program files are rather lengthy we have zipped them and put them on the internet on our homepage at: `http://mac2-csg.teknikum.uu.se/ proc9801/`.

5.1 Java files

ProcGUI.java This is the file containing the code for the actual interface. The panels, tabs, buttons, slidebars etc. are all defined here together with their respective actions. There are also "action listeners", and "key listeners" defined in this file to detect when buttons are depressed in the interface or keys are struck on the keyboard. When this is detected appropriate action is taken. These actions are defined as procedures and for the most part consist of sending packages to the Matlab process and/or enabling or disabling buttons and panels in the interface, or changing of preset values on the interface.

AWT_Container.java The purpose of this class is to provide a kind of global variables for the different classes to use for communication purposes.

Listener.java The main objective of this class is to receive messages sent by the Matlab side. These messages arrive once every sample and every message consists of a name and four matrices with information. For example after receiving some of the messages this class prints an appropriate text in the interface message display and/or disables/enables different functions in the GUI.

RTView.java This file was originally supplied to us in order to handle the scrolling display for the reference and measured signals. However it has been heavily improved by us for example with the ability to show a non-flickering grid resembling that of an oscilloscope behind the curves.

RTViewCommon.java Similar to the *RTView.java* file but used to produce a still diagram for the snapshot feature.

RTViewMotor.java This class plots the motorsignals against each other as mentioned earlier.

Alarmpanel.java This class is used to handle the alarms in the system.

UDP_Packet.java This class is Used to handle the UDP communication.

5.2 Matlab files

regulator.m This file contains all of the code needed for the regulation, alarming and communication on the Matlab side except for the code used by the separate adapting process which is contained in *subregulator.m*.

subregulator.m This file contains the code for the subprocess that is constantly trying to adapt to the dynamics of the system. It communicates with the other process on the Matlab side to exchange values e.g. initialization, reset and current values.

6 Problems encountered during the project and solutions found

6.1 General problems and our solutions

- We had some problem with the fact that an ethernet is a fairly lousy connection to use for real time traffic. The reason for this is that when two stations try to transmit on the ethernet at the same time, they will both stop sending when they notice that there has been a collision. After that they will wait for a random(!) amount of time before trying to retransmit their messages. This random amount of time is increased after each unsuccesful attempt to retransmit and the result is that waiting times can be very long and unpredictable. We tried to reduce traffic as much as possible by gathering some packets up in a bundle and sending them as a whole from the Matlab side to the Java side. The Java program then extracts the individual messages from this packet. In the communication between the main Matlab process and the subprocess we chose not to collect packets in a bundle because there is only one packet that is frequently occuring (once every sample).
- It was a bit difficult to adjust the PID:s for the best possible regulation as required in the specification. We did not get good results until we introduced nine different work voltage limits. When the user chooses to adjust the PID parameters manually the parameters are taken directly from the interface. However, when PID control is set to automatic, the system determines which of the nine sets of values to use depending on the reference signal value.

6.2 The Java side

Drawing a grid behind the diagrams showed to be quite a problem since the amount of computing time per sample is a fairly limited resource. The problem was solved by using a clever algorithm that only draws the necessary pieces of the grid in each sample. The algorithm is also double buffering to achieve a flickerless image.

6.3 The Matlab side

- We had a little trouble adjusting the alarms correctly so that they would not alarm too easily but still be safe enough. It took some experimenting to get the best adjustment and to get the system to alarm for all sorts of erraneous behaviour.
- The adaptive regulation took some time to adjust correctly after a change of regulator type. We solved this problem by having an extra Matlab process (subregulator.m) that adapts the whole time. The adaption is turned off when good regulation has been achieved.

- It was a bit difficult to get the risetime and overshoot calculations to cover all possible sorts of signal shapes when given a step as reference. For example we now handle the case in which a signal oscillates a bit before it reaches its ninety percent value.
- A problem that occurred when we added a real time plot on the Matlab side was that the computer did not have enough capacity to update everything in the time between samples. This problem can however be solved by starting a separate Matlab process for handling plots.

7 Users' manual for the control system

7.1 Starting up the system

1. Start the *subregulator.m* Matlab process under ordinary *Matlab* on the Linux machine.
2. Start the *regulator.m* Matlab process under *Matlab* with real time extensions (*procmatlab*) on the Linux machine.
3. Start the *ProcGUI.class* Java program on the Windows NT machine either by surfing into our homepage at `http://mac2-csg.teknikum.uu.se/ proc9801/` or running the class from *Symantec Visual Cafe*.

7.2 Running the system

When the system starts up the default regulator started is the PID for both tension and velocity. See figure (4) for an illustration of how the system looks just after startup.

In regular mode the user can do the following from the GUI:

1. The reference signals for tension and velocity can be altered separately by moving the appropriate slidebar or typing in a value as can be seen in figure (5).

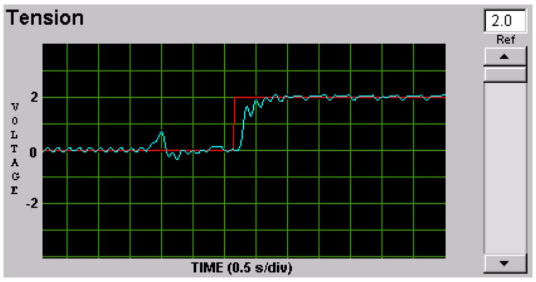


Figure 5: *Reference signal controls.*

2. Regulators can be changed separately for the tension and velocity by clicking on the appropriate tab on the panel as can be seen in figure (6).

Figure 6: *Regulator tabpanel.*

3. A step can be performed for either the tension or the velocity by pressing a button. The height for a step can be filled in next to the button as seen in figure (7). If the amplitude of the step is too large the step will be halted at the maximum (or minimum) voltage.

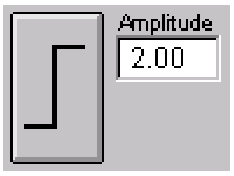


Figure 7: *Step adjustment.*

4. A ramp can be performed for either the tension or the velocity by pressing a button. The height and slope for a ramp can be filled in next to the button as seen in figure (8). If the amplitude of the ramp is too large the ramp will be halted at the maximum (or minimum) voltage.

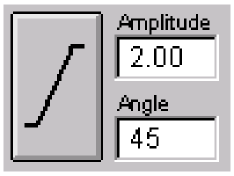


Figure 8: *Ramp adjustment.*

5. A snapshot of the tension or the velocity graph can be taken for a user defined period of time. The period may however not be greater than one full screen width (6.4 seconds) since no more values are buffered. This snapshot is presented on a seperate display as seen in figure (9).

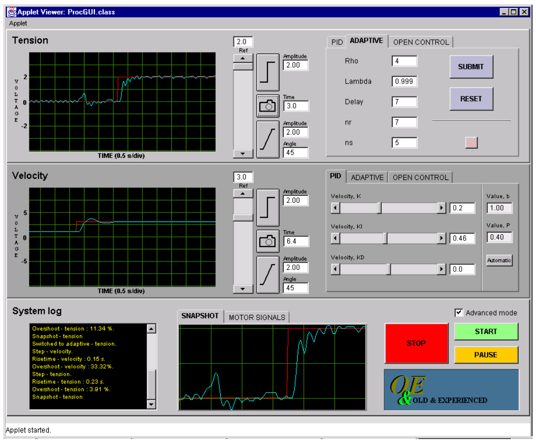


Figure 9: *Snapshot.*

6. The status window can be scrolled back if the user wishes to check the complete history of the process as can be seen in figure (10).

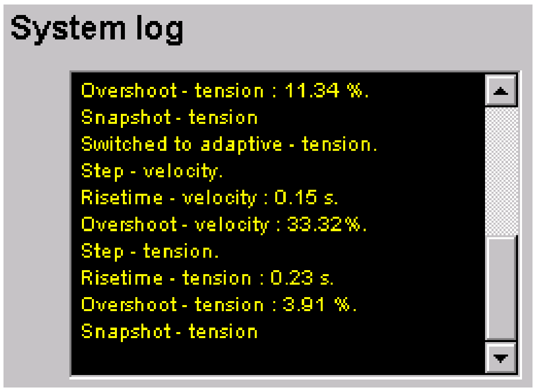


Figure 10: *Status window.*

7. In the mid bottom of the screen, in the tab panel, a choice can be made between plotting the motor signals (U_1 and U_2) or the latest snapshot taken. When one of the snapshot buttons is pressed this tab panel is automatically switched to showing the snapshot. See figure (11) for a picture of the motor signal plot.

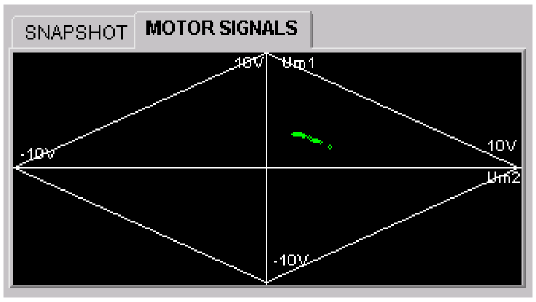


Figure 11: *Motor signals.*

8. The user can choose to stop, pause or start the process by pressing the buttons in the lower right corner as can be seen in figure (12).

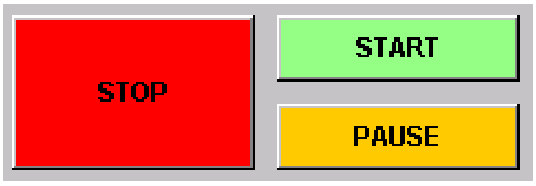


Figure 12: *Stop, start and pause buttons.*

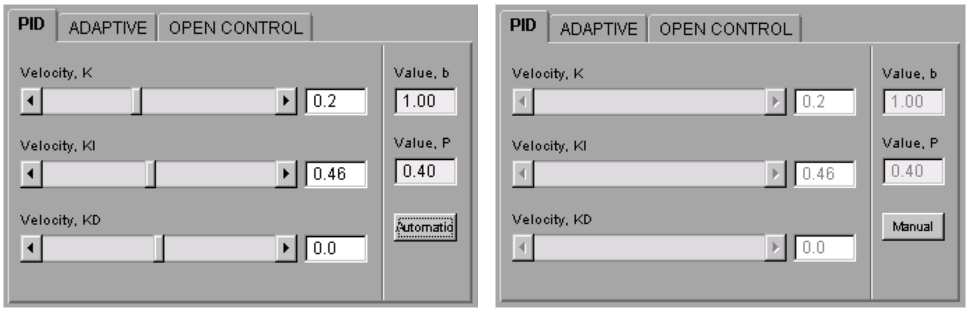
9. The logotype in the lower right corner of the GUI has no special purpose but you can see it in figure (13).

Figure 13: *Logotype of the Old and Experienced group.*

In advanced mode, which is selected by marking the *Advanced mode* checkbox (see figure(14)), the user can in addition to the above mentioned also perform the following from the Java GUI:

Figure 14: *Advanced mode checkbox.*

1. In the PID panel, the following values can be set: K, KI, KD, P and b. See figure (15) for an example of this.



(a) PID manual (b) PID automatic

Figure 15: *PID panel.*

2. In the adaptive control panel, the following values can be set: rho, lambda, delay and polynomial orders. See figure (16) for an example of this.

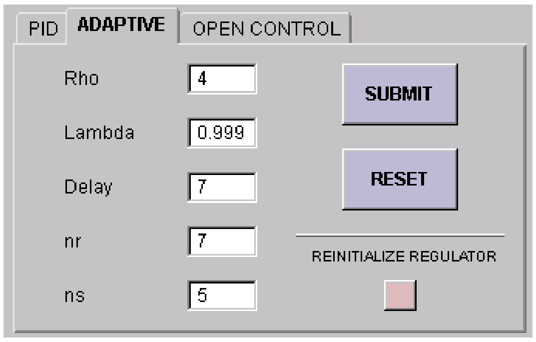


Figure 16: *Adaptive control panel.*

3. In the open control panel, no values can be set as can be seen in figure (17).

Figure 17: *Open control panel.*

7.3 Shutting down the system

1. Press the **STOP** button in the Java GUI to terminate the main Matlab process (*regulator.m*).
2. Exit the Java GUI interface.

8 Results and discussion

Our group is very satisfied with the finished regulator and its performance. We also think that our GUI is very easy to use and still provides the user with a complete picture and feel of the system. This course has been a great learning experience when it comes to modern process control. The only drawback is that the amount of work is not in proportion to the number of points on the course. The course ought to be worth at least 6 points.

A Error plots

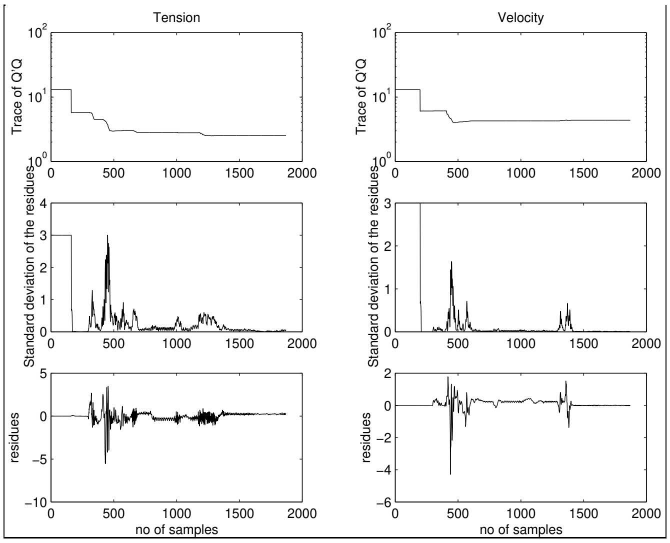


Figure 18: *Error plots showing no windup but the standard deviation limit is chosen much too small.*

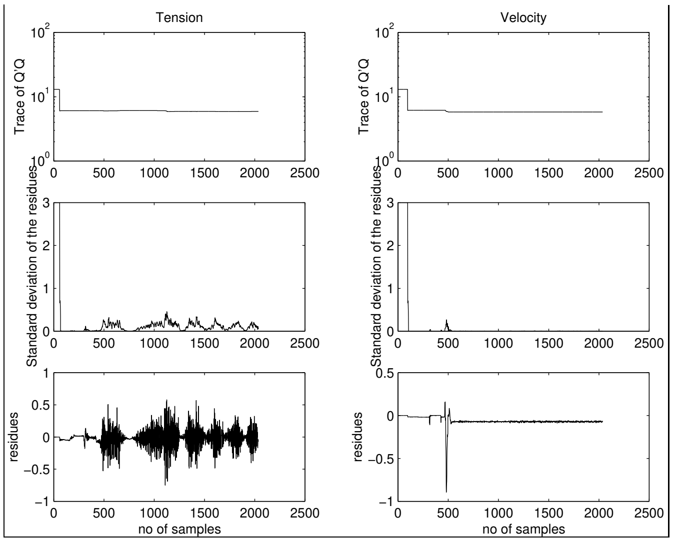


Figure 19: *Error plots showing the perfect standard deviation limit with no windup at all.*

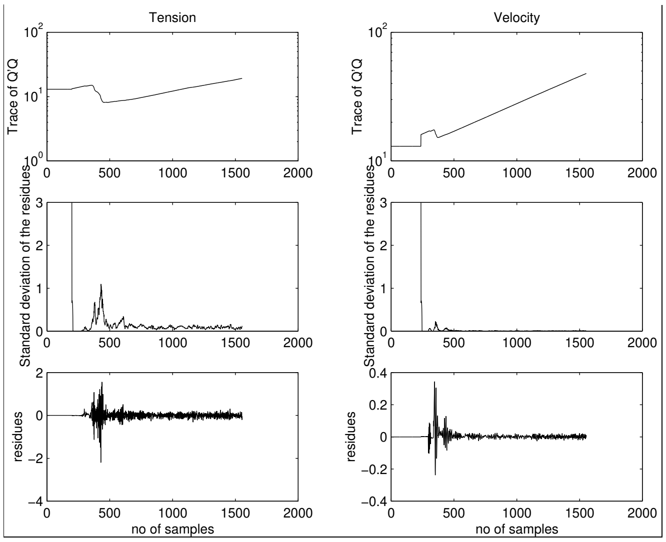


Figure 20: *Error plots showing the case where the standard deviation limit is choosen too small i.e. windup in the system.*