ISSN 0280-5316
ISRN LUTFD2/TFRT--5721--SE

Feedback Control of Thermoacoustic Instability Using Acoustic Actuator

Daniel Mijic
Calle Stödberg

Department of Automatic Control
Lund Institute of Technology
February 2004

Department of Automatic Control **Lund Institute of Technology** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER THESIS *Date of issue* February 2004 *Document Number* ISRN LUTFD2/TFRT--5721--SE
Author(s) Daniel Mijic and Calle Stödberg	*Supervisor* Rolf Johansson LTH, Lund *Sponsoring organization*

Title and subtitle

Feedback Control of Thermoacoustic Instability Using Acoustic Actuator (Akustisk reglering av termoakustisk instabilitet).

Abstract

Thermoacoustic instability is a phenomenom that appears in several technical applications, for instance rocket and jet engines, gas turbines, waste generators and industrial burners where it can cause heavy damage to the systems. As with most instabilities, they are undesired and this thesis´ purpose is to identify and control a Rijke tube process (which has been used to stimulate the thermoacoustic instabilities) with the aid of microphones and a loudspeaker. A major part of the work has been done on setting up such a process, considering for instance the necessary high sample rates and make future experiments possible and also to show that the process can be controlled with very simple means in a very simple environment.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 45	*Recipient's notes*
Security classification		

Contents

1 Introduction **4**
1.1 Motivation . . . 4
1.2 Objectives . . . 4
1.3 Method . . . 4
1.4 Rijke Tube . . . 4
1.5 Thermoacoustic Instability . . . 5
1.6 Active Control of Thermoacoustic Instability . . . 6

2 The Laboratory Equipment **7**
2.1 The Flameholder . . . 7
2.2 The Software . . . 8
2.2.1 RTAI . . . 8
2.2.2 COMEDI . . . 8

3 System Identification **9**
3.1 Input Signal Selection . . . 9
3.2 Nonparametric Model . . . 11
3.3 Model Structure . . . 11
3.4 Parametric Model . . . 13
3.5 Validation . . . 16

4 Control Design **18**
4.1 Sample Rate Decision . . . 18
4.2 Stability Analysis . . . 18
4.3 Controller Design . . . 19
4.4 Results . . . 20
4.5 Conclusions . . . 20

5 Acknowledgements **21**

A RTAI Structures **23**
A.1 getFrequencyScheme . . . 23
A.2 controllerScheme . . . 23

B C-files **23**
B.1 getfrequency.c . . . 23
B.2 logger.c . . . 25
B.3 generate_sin.c . . . 26
B.4 p_shift_controller.c . . . 28
B.5 control_logger.c . . . 30

C Header Files **31**
C.1 constants.h . . . 31

D Matlab Files **32**
D.1 getFrequency.m . . . 32
D.2 getParModel.m . . . 32
D.3 getNonParModel.m . . . 34
D.4 validateModels.m . . . 34

D.5 iScript.m 35

E System Identification **35**

F Makefile **38**

F.1 Makefile 38

G Run Scripts **40**

G.1 startup 40

G.2 removeM 40

G.3 run 40

H Amplifier and Loudspeaker **41**

1 Introduction

1.1 Motivation

To understand and to be able to control thermoacoustic instabilities is necessary in order to efficiently eliminate heavy pressure oscillations that disturb and, in worst case, damage applications such as gas turbines, jet and rocket engines.

Instability in combustion processes also leads to increased pollutant emission which definitely motivate the importance of this work. Another problem is the noise that unstable combustion processes often create.

1.2 Objectives

The purpose of this work has been to set up and identify a flame heated Rijke tube process in order to study and control thermoacoustic instability phenomena by using an acoustic actuator such as an loudspeaker and microphones as sensors.

1.3 Method

This thesis includes the developing of a Rijke tube connected with hardware and software, system identification of the process and control theory. The work was performed at Lund institute of Technology (LTH). The software was programmed using C and MATLAB for experiments and identification.

1.4 Rijke Tube

One of the most common ways to experience the phenomena and effects of thermoacoustic instability is by studying a classic Rijke tube. It typically consists of a vertical pipe with a gauze, often located in the lower half of the tube (not necessarily though), that is heated either electrically or with a flame. Due to the combination of heated air leading to mean air flow and pressure oscillations the tube will generate a high-intensity tone when the gauze has reached a sufficiently high temperature. Rijke [4] discovered that oscillations were most intensive when the heating source was placed one quarter of the tube length from the lower tube end.

Rayleigh proposed a criterion saying “If heat be periodically communicated to, and abstracted from, a mass of air vibrating in a cylinder bounded by a piston, the effect produced will depend upon the phase of the vibration at which the transfer of heat takes place. If heat be given to the air at the moment of greatest condensation or to be taken from it at the moment of greatest rarefaction, the vibration is encouraged. On the other hand, if heat be given at the moment of greatest rarefaction, or abstracted at the moment of greatest condensation, the vibration is discouraged”.

A glass Rijke tube, as described above, has been used throughout the project. The heating source has been a regular Bunzen burner with a metal net as a flameholder. A problem with a conventional Rijke tube may be the difficulty to control system parameters, such as heating source location, air flow rates and heat release, in an exact way. An electrically [3] heated Rijke tube with a blower providing the air flow would overcome these problems easier but it does not allow studies of the changes of the flame's appearance, which has been one of the goals throughout the project. Further, to prove that the thermoacoustic instabilities could be controlled with, so far, very simple means in a very simple environment.

1.5 Thermoacoustic Instability

Thermoacoustic instability is a phenomenon that appears in several technical applications. A few examples are rocket and jet engines, gas turbines, waste generators and industrial burners. Thermoacoustic instability can simplified be described as the positive coupling between unsteady heat release and acoustics. A diagram of this coupling can be viewed in Fig. 1.

Combustion processes often consist of a combustion chamber, a flame and a fuel/air mixture. The flame, which is mounted in the combustion chamber ignites the fuel/air mixture and energy from the gas is released and can be used by application.

In most cases thermoacoustic instability disturbs the application. Heavy pressure perturbations can damage the combustion chamber and the sound generated is also undesirable. Components or the whole chamber can be destroyed or even melt if the heat released from the system is to high. A way to decrease the temperature in a combustion chamber is to premix the fuel and air before it is ignited. A lower temperature leads to less NO_x emissions. However when the fuel and air is premixed the heat release is more concentrated, which decreases the stability margin and often triggers the thermoacoustic instability. There are some applications were thermoacoustic instability is used to get a higher efficiency. Examples of such applications are special types of pulse waste generators.

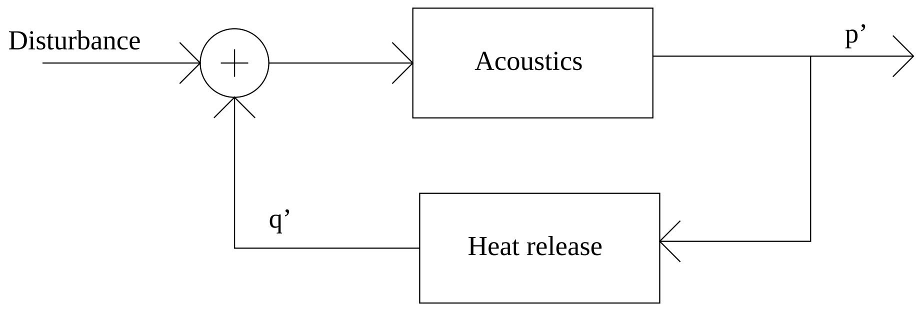

Figure 1: Thermal instability loop.

1.6 Active Control of Thermoacoustic Instability

The heat released from the flame in a combustion system can be actuated either by changing the fuel/air mixture or by manipulating the structure of the flame. Velocity and acceleration of air that is generated by the acoustic field can disturb the appearance of the flame. In this way the heat released is disturbed.

There are several ways to active control a combustion instability. Two categories of actuators that can be used are flow sources and heat sources. A loudspeaker can be used as a flow source and some kind of fuel injector as heat source. In a Rijke Tube it is also theoretically possible to control/change the flame location. Controlling the process using the flame location as an actuator is not fast enough. A flow source changes the velocity of the flow and impacts both the heat release and the acoustics in a direct way. A heat source changes the heat release. Indirect this increased/decreased heat release impacts the acoustics. As can be seen neither of these alternatives only affects the acoustics. In this thesis a loudspeaker was used as actuator.

2 The Laboratory Equipment

The laboratory equipment consisted of a 750 mm glass pipe with an inner diameter of 65 mm. A 5 W loudspeaker was used as actuator (App. H).

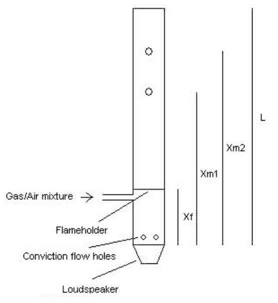

Figure 2: Drawing of the Rijke tube. L = 750 mm is the length of the glass tube, Xf = L/4 location of the flameholder, Xm1 = L2/3 location of microphone 1, Xm2 = L5/6 location of microphone 2.

2.1 The Flameholder

What was discovered very early during the work with the Rijke tube was that the flameholder plays a very important role if there is to be any sound at all in the tube.

A flameholder, or flame controller, is often placed above regular burners. It's purpose is to prevent the flame from spreading into the burner and to shape the flame. It is in the simplest case just a small metal net which is placed above the flame .

At the first attempts with the Rijke tube the burner had a small net which was placed directly, almost flat, on the burner. The result was a quiet tube. After trying different set-ups mainly changing the position of the glass tube according to the burner with no improvements regarding the sound (which was not there) a new flameholder was given a try. In fact, it was the same flameholder as before but it was a little reshaped in a way that allowed to it be placed a little bit above the burner, like a balloon, instead of lying directly on the burner. The glass pipe was merely put on before a loud and clear tone could be heard.

Successful attempts with a flameholder shaped like circular lid have also been performed and this kind of flameholder has been used for the final version of the Rijke tube.

Conclusions are that the flameholder must be shaped in such a way that the position of the burner makes it possible to create a flame big enough (or actually concentrating enough heat in one place) to heat enough air (flowing through the pipe) fast enough which is essential for the unsteady heat release and pressure oscillations that lead to thermoacoustic instability.

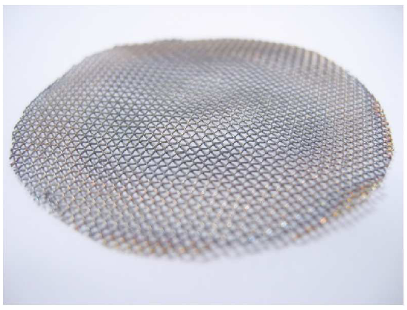

Figure 3: The flameholder.

2.2 The Software

2.2.1 RTAI

The human ear can hear sound which approximately has a frequency in the range 20 Hz - 20 kHz. The frequency of the sound generated by the Rijke Tube was determined. Microphones were used to sensor the pressure vibrations. Sampling of such high frequencies requires a hard real time system, which is not interrupted by tasks that are not used in the control system. Linux Real Time Application Interface (from here on RTAI) was used to get a hard real time system.

RTAI is a real time kernel, which runs besides the ordinary Linux kernel. The real time kernel controls the interrupt handling. It decides whether an interrupt should be passed to Linux or not. Linux is prevented from disable interrupts. In this way Linux can not block interrupts or stop itself from being pre-empted. Linux tasks are only allowed to run when there are no real time tasks that wants to run. RTAI treats Linux as a task with the lowest priority.

RTAI also provides scheduling and gives access to real time services. Examples of such services are FIFOs, shared memory, POSIX thread compatibility and the ability to use Real Time Linux (RTLX). RTAI applications must run in Linux kernel memory space. Modules that are loaded into kernel space are used to access the kernel in Linux. RTAI applications are implemented as modules and can access the kernel memory. RTAI is implemented in the C programming language and applications using it are often programmed in C. Using RTAI gives the possibility to implement a hard real time system and also use the ordinary Linux services e.g. the file and window systems.

2.2.2 COMEDI

The PC was equipped with an Adventech 1711 AD/DA card. It has 16 analog inputs(12 bits), 2 analog outputs, 16 digital inputs and 2 digital outputs. The card was connected to the microphones and to the loudspeaker via analog inputs and outputs. To be able to communicate with the card from the PC some kind of device driver is needed. For AD/DA cards (also called DAQ - Data Acquisition Card) an API (Application Programming Interface) called

COMEDI was available. COMEDI which is a Linux Control And Measurement Device Interface is used to access DAQ cards. COMEDI contains a collection of device drivers for DAQ cards. Using COMEDI gives the possibility to change to another AD/DA card without rewriting the software. The API can also be accessed from the Linux kernel, via a module, which makes it possible to communicate to the card from real-time applications. COMEDI can be used from RTAI applications.

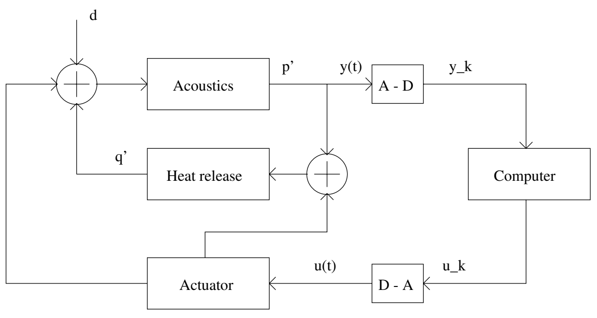

Figure 4: Scheme over the control system. The actuator affects the acoustics in a direct way and an indirect way, via the increased/decreased heat release.

3 System Identification

A good way to get more knowledge about the process is to use some kind of system identification methods. The combustion dynamics of the process are highly nonlinear, which normally makes the identification harder. The system parameters and especially the pressure end up in a stable limit cycle. One way to use system identification on the combustion process is to linearize the system in the neighbourhood of this limit cycle. The identification procedure includes a choice of suitable input signal, data examination, model structure selection, model estimation and validation.

3.1 Input Signal Selection

The end-mounted speaker is used to generate the input pressure to the process. This pressure will affect the states of the process and the output pressure generated by the process will be measured. Hence, both the input and output signal of the process is considered as pressure. In the identification procedure the amplifier, loudspeaker and microphones are included in the process. The output current of the amplifier is proportional to the input current. The pressure generated by the loudspeaker is proportional to the input current under the constraint that the second derivative of the input current is not equal to zero. This can be written as

$$u_l'' \neq 0 \Rightarrow p_l = k_l u_l \tag{1}$$

where u_l, p_l and k_l are the input current, output pressure and a constant of the loudspeaker, respectively. Finally, the output current of the microphone is proportional to the output pressure of the process.

The identification gives a better model at the frequencies where the input signal contains much energy, the input signal has good excitation at these frequencies. A pseudo random binary sequence (PRBS) was used to create disturbances to the input signal used for identification. The sampling interval was chosen to 10 kHz and the data where collected under 10 seconds. 5 seconds of the data was used for identification and 5 seconds for validation.

It was possible to add the PRBS signal to the phase, amplitude or frequency of the input current. Combinations of these PRBS signals were also tested. Different input signals were generated, see App. E. Finally the input signal named $i10\_1.mat$ was used for the identification. This signal has good excitation in the frequency range, this can be seen in Fig. 5.

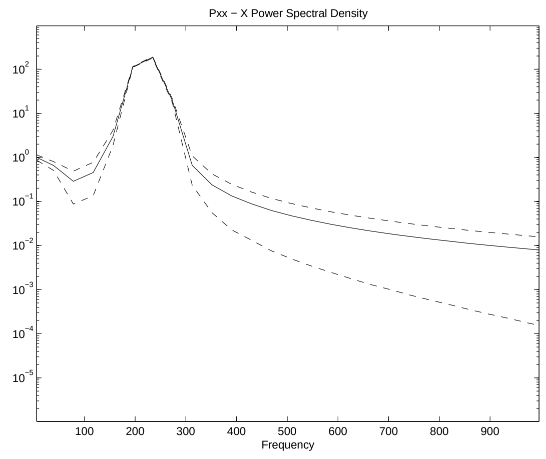

Figure 5: Power spectral density of the input signal.

A coherence spectrum [7] is a test of linearity between the input and output of a process. The spectrum shows for which frequencies there is a linear dependency between the input and output signals. A value close to 1 indicates a perfect linear dependency. The coherence spectrum for the input and output signals, used for identification can be viewed in Fig. 6. The coherence spectrum shows that there is a linear dependency between the input and output signal in the frequency range 150 - 280 Hz. This indicates that it should be possible to get an accurate linear model of the process in this range.

When performing system identification it is an advantage if the signal to noise ratio S/N is high. One way to measure the signal to noise ratio is to use the equation Eq. (2) below.

$$S/N = \frac{std(y_{no_i nput})}{std(y_{PRBSinput})} \tag{2}$$

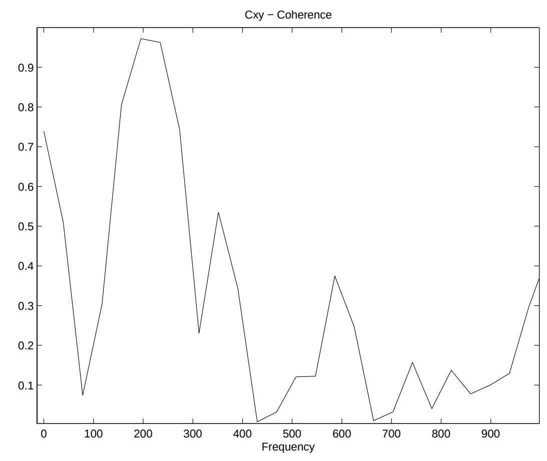

Figure 6: Coherence spectrum.

where std is the standard deviation of the signal, y_{no_input} is the output when no input is used and $y_{PRBSinput}$ is the output when the PRBS signal is used as input. When the $i10\_1.mat$ data was used as input the signal to noise ratio was approximately 2.

3.2 Nonparametric Model

A nonparametric model of the process was identified using spectral analysis. A great advantage of nonparametric models is that they need no specification for model structure and model order. A bode diagram, displaying the gain (Fig. 7) and phase (Fig. 8) of the process transfer function was generated by spectral analysis, using the MATLAB function spectrum. The amplitude diagram indicates that the process is a damped oscillator where the gain has a maximum around 250 Hz, which was expected.

3.3 Model Structure

When deciding the model structure there is some structures that can be used. The difference between these lie mainly in how the external noise influences the system. An ARMAX model (autoregressive moving average with exogenous input) was used to model the system. The ARMAX model has the following structure

$$A(q)y(t) = B(q)u(t - n_k) + C(q)e(t) \tag{3}$$

where n_k is the time delay and A, B and C are polynomials and where q is the forward shift operator, y(t) is the output signal, u(t) input signal and e(t) white noise.

There are two main parameters in the ARMAX model that has to be decided.

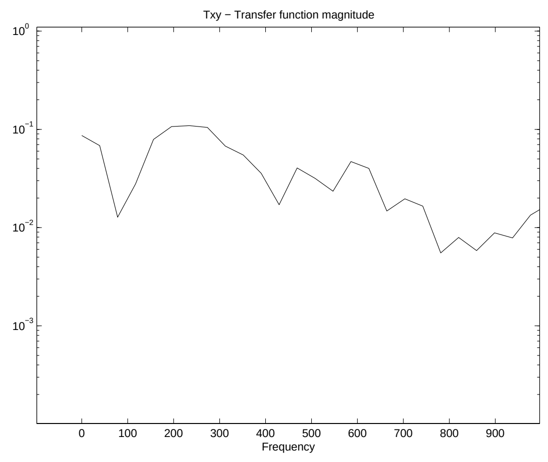

Figure 7: Bode amplitude diagram of the process transfer function. Generated by the MATLAB function spectrum.

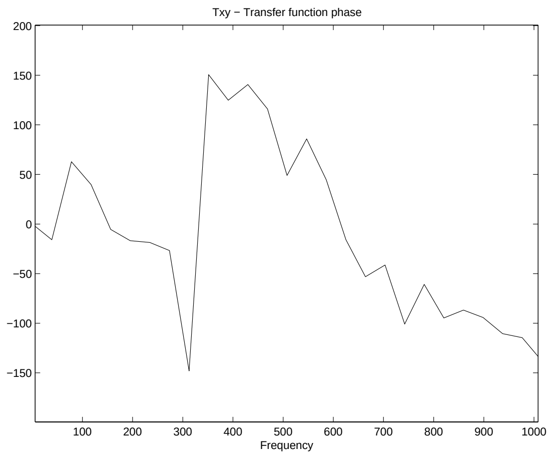

Figure 8: Bode phase diagram of the process transfer function. Generated by the MATLAB function spectrum.

First there is order of the system, i.e. the degree of the A, B and C polynomials. Second there is the time delay of the system.

3.4 Parametric Model

Using some kind of validation criterion is one way to determine which ARMAX model that best describes the combustion system. Correlation between simulated and measured output is a good measure of how well the model manages to imitate the combustion process. Data used for identification and verification was not the same when correlation analysis was used. If the same data was used it would be easy to find an accurate model. This kind of correlation analysis was used to find an accurate model of the system.

The MATLAB function armax was used for the identification. Early tests showed that a low order of the system is enough. A higher degree than three of the A-polynomial was not necessary to test.

The distance d between the end-mounted loudspeaker and the microphone used to sensor the output is a key parameter of the system. The microphone was placed on the distance d = L2/3 from the loudspeaker. L = 0.75 ⇒ d = 0.75·2/3 = 0.5 m . The speed of sound v is approximately 341 m/s. Hence, the time t it takes for the pressure generated by the loudspeaker to reach the microphone is t = d/v = 0.5/341 = 0.0015s . This time, result in a time delay of the combustion system that is equal or greater than 0.0015 s, this is the same as 15 samples. As mentioned earlier the loudspeaker will impact the system in a direct way, via the pressure. It will also impact the system in an indirect way, via increased or decreased heat release. One way to determine the time delay of the system is to examine the correlation function between the input and output signal. The correlation function of the system was generated by the MATLAB function cra, see Fig. 9. The picture shows that there is a linear dependency between the input signal u and an output signal y that is delayed approximately 25 samples.

To determine the model order A MATLAB-script *getParModel.m*, App. D.2, was implemented. The script tested the correlation for different ARMAX models. Two different time delays was used, 15 and 25 samples. The result was plotted and can be viewed in the Fig. 10. The lowest order model that gives high correlation was an [aDegree bDegree cDegree timeDelay] = [2 2 1 25] model, where aDegree, bDegree and cDegree are the orders of the A, B and C polynomials, respectively, and timeDelay is the time delay of the system. The correlation coefficient for the [2 2 1 25] model is 0.9908 and the model has the following polynomials

$$A(q) = 1 - 1.938q^{-1} + 0.9644q^{-2} \tag{4}$$

$$B(q) = -0.03792q^{-25} - 0.01311q^{-26} \tag{5}$$

$$C(q) = 1 - 0.6379q^{-1} \tag{6}$$

Information that can be studied in the Fig. 10 is the difference between the two time delays 15 and 25 samples. The same figure with higher resolution, Fig.11, clearly shows that 25 samples time delay is to prefer against 15 samples.

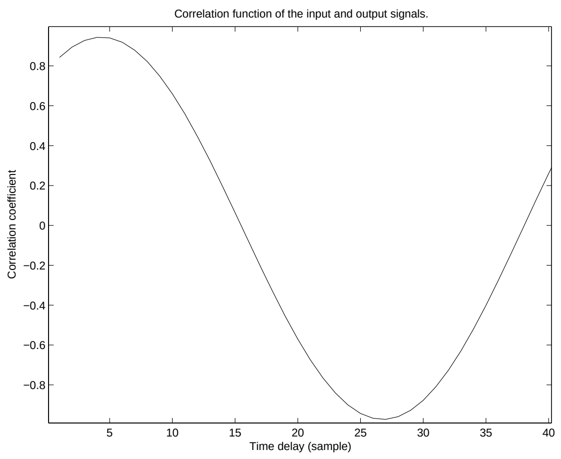

Figure 9: Correlation function of the input and output signals.

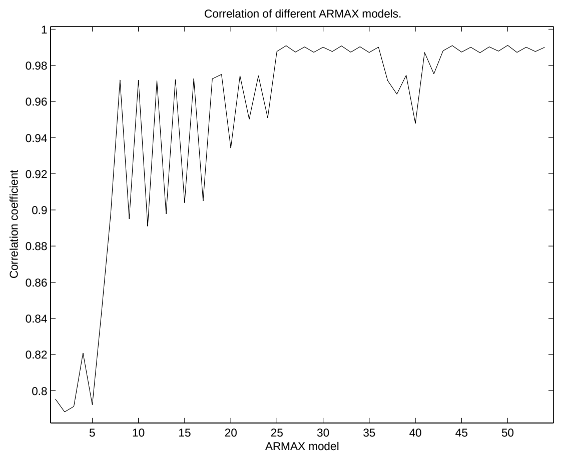

Figure 10: Correlation coefficient of different ARMAX models.

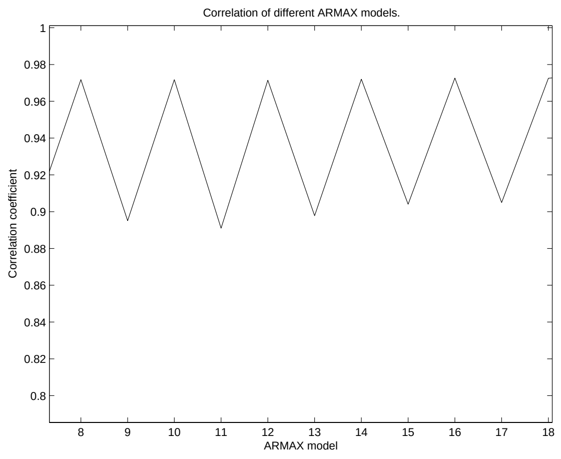

Figure 11: Correlation coefficient of different ARMAX models. Even number on the x axis represent a model using 25 samples time delay and odd number 15 samples time delay.

If the ARMAX model [2 2 1 timeDelay] was chosen it could be interesting to see how different time delay describes the system. The MATLAB-script *getParModel.m*, D.2, was also examining this idea. Fig. 12 shows the correlation of the ARMAX model [2 2 1 timeDelay] as a function of the timeDelay variable. A time delay greater than 15 gives an accurate model.

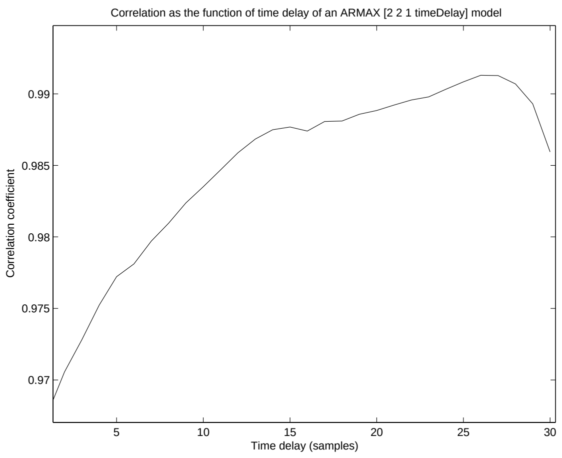

Figure 12: Correlation as the function of time delay for an ARMAX [2 2 1 timeDelay] model.

3.5 Validation

A parametric model can be validated in many different ways. The ARMAX model, that was determined in the previous section was validated using simulation and cross validation. The simulated and measured outputs can be viewed in Fig. 13. By using the two MATLAB functions th2zp and zpplot it was possible to plot Fig. 14 the zeros and poles of the discrete ARMAX [2 2 1 25] process. 14

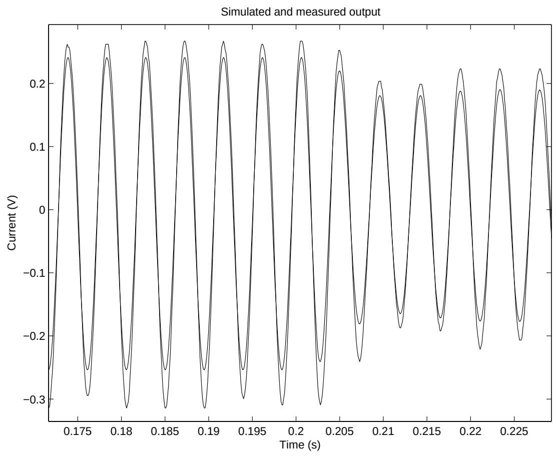

Figure 13: Simulated and measured output.

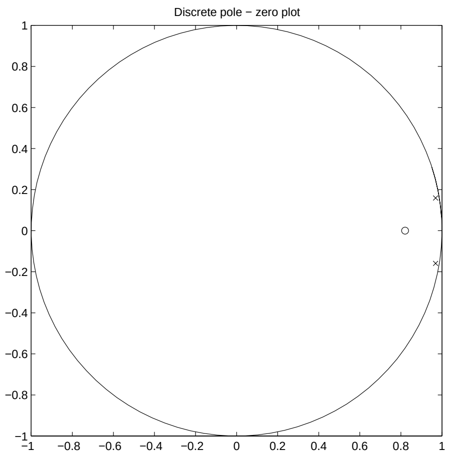

Figure 14: Pole zero plot of the discrete [2 2 1 25] ARMAX model.

Another way to validate the model is to look at the power spectrum of the residuals when comparing measured and simulated output. This spectrum can be shown in the Fig. 15. An optimal power spectrum would be zero at all frequencies.

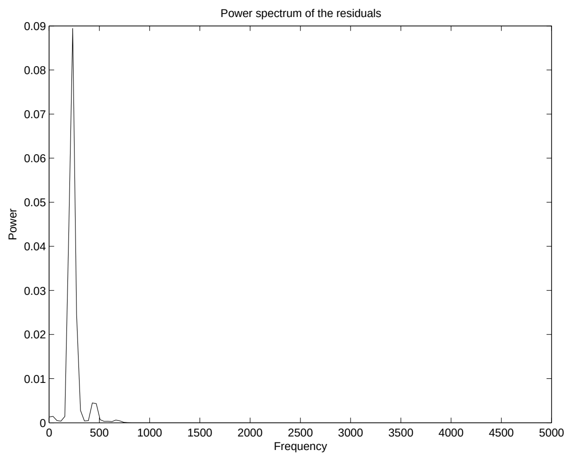

Figure 15: Power spectrum of the residuals.

The last validation used was to compare the bode diagram (Fig. 16) of the parametric model with the bode diagram received by the nonparametric model. The amplitude diagram matches the amplitude diagram of the nonparametric model.

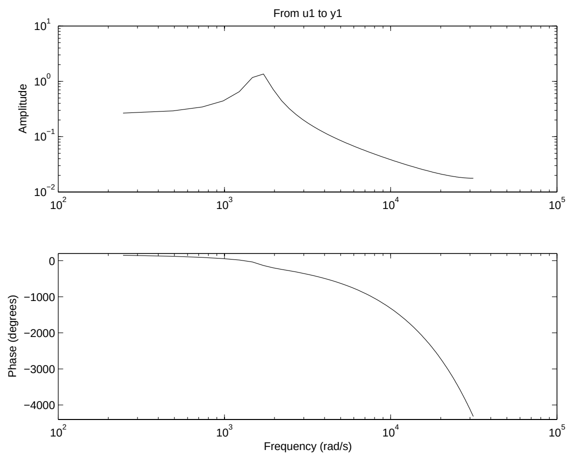

Figure 16: Bode diagram of the [2 2 1 25] ARMAX model.

4 Control Design

4.1 Sample Rate Decision

When the sampling rate of the control system was to be decided it was an advantage if the unstable frequency of the tube could be determined. The tube had the length 0.75 m. When instability occurs in the tube there is a standing wave in the tube. The wave has the wavelength $\lambda = 0.75 \cdot 2 = 1.5m$. The speed of sound v is approximately 341 m/s. The frequency of the standing wave is: $f = v/\lambda = 341/1.5 \approx 227Hz$.

The speed of sound depends on the medium. A RTAI application used to measure the frequency of the sound was implemented to ensure that the calculation over the unstable frequency was valid. (App. B.1, B.2, A.1). The application read the inputs from the two microphones and wrote the samples to two files, one file for each one of the microphones. The sampling rate used by the application was 10 kHz. Frequencies of the sound that are below the Nyquist frequency $f_N = f_s/2 = 10000/2 = 5kHz$ could be measured by the application (see [1]).

The two files, containing 50000 samples collected during five seconds, generated by the program was read into the MATLAB (App. D.1) memory space. A plot of the samples can be viewed in Fig. 4.1. As can be seen in the plot the frequency of the sound is approximately 250 Hz. Using the theory of aliasing and the Nyquist frequency it is enough to use a sampling frequency $f_s = 2 \cdot 250 = 500Hz$ to avoid aliasing of the sound. To ensure that no frequencies were lost in the control loop a higher frequency than 500 Hz was used.

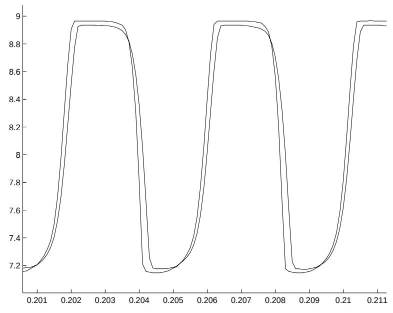

Figure 17: Samples from the two microphones.

4.2 Stability Analysis

An equation over the stability criterion for combustion process can be written (see [2][5]) as:

$$\frac{\delta}{\delta t}\int_0^L e' dx = \frac{\gamma - 1}{\overline{\rho c^2}}\int_0^\tau \int_0^L p'q' dx dt - \Delta_L(E') - \Phi, \tag{7}$$

where e' is the acoustic energy, E the acoustic energy flux and Φ energy dissipation. p' and q' is the pressure and the heat release perturbation. γ, ρ and c are the specific heat ratio, density and the speed of sound. x and t stands for distance and time. Δ_L is the difference over the combustion length L. The mean flow is neglected in the equation above. A combustion system is stable if the value of the equation is less than or equal to zero. The system is unstable when the heat release fluctuates in phase with the pressure perturbation and the energy losses are smaller than the generated energy.

The acoustic energy field stores the internal energy of a combustion system. The flame and an actuator will give or take energy from the acoustic energy field. A loudspeaker used as actuator will affect the internal energy in a direct and an indirect way. Direct by generating pressure via its acceleration and indirect by adding air flow to the flame, which affects the heat release. Using this theory, Eq.(7) can be written as

$$\frac{\delta}{\delta t}\int_0^L e' dx = W_f + W_d = W_q + W_i + W_d, \tag{8}$$

where W_f is the work exchange between the acoustic field and the flame, W_d between loudspeaker and acoustic field directly, W_i between the loudspeaker and the flame and W_q between the flame and the acoustic field. When no actuator is used to control the process $W_d = W_i = 0$. For an unstable system the energy is increasing and $W_f = W_q > 0$. When an actuator is used the criterion for stability is $W_i + W_d <$ -W_q. The above equation 8 shows that a loudspeaker can affect the stability criterion. In a similar way it is possible to show that other actuators can be used.

4.3 Controller Design

Thermoacoustic instability is often modelled as the Rayleigh's criterion, which can be written as[5]:

$$\int_0^\tau \int_0^V p'(x,t)q'(x,t)dvdt > \int_0^\tau \int_0^V \Phi(x,t)dvdt, \tag{9}$$

where p' is pressure perturbations, q' perturbations in the heat release rate, Φ the wave energy dissipation, τ period of oscillation and V is the combustor volume. The LHS of the criterion describes the mechanical energy added to the oscillations by the heat release. Rayleighs criterium can be derived from Eq. (7)

If the inequality Eq. (9) is satisfied, instability will be encouraged. The wave energy dissipation Φ is often very small and can be neglected. The criterion shows that instability will occur if the pressure and heat release perturbations oscillates in phase.

The limit cycle behaviour of the system indicates that the heat release and pressure states oscillate at same frequency. The sign of the time integral Eq. (9) will then depend on the phase between the heat release and pressure perturbations. It has been shown [5] that the integral has a maximum when $\frac{\tau_0}{\tau} =$ 0, 1, 2, 3... , and a minimum when $\frac{\tau_0}{\tau} = \frac{1}{2}, \frac{3}{2}, \frac{5}{2}$...

Using the stability criterion stated above it was possible to implement a controller (App. B.4) that stabilized the combustion process. The period $\tau =$

0.0040s of the process limit cycle has previously (Ref. 4.1) been determined. The controller reads the output value y(t) (pressure) from the process and calculates a control signal u(t) using the Eq. (10).

$$\begin{aligned} y(t) + u(t) &= y(t + \varphi) \Leftrightarrow \\ \sin(\omega t) + u(t) &= \sin(\omega t + \varphi) \Leftrightarrow \\ u(t) &= \sin(\omega t + \varphi) - \sin(\omega t) \end{aligned} \tag{10}$$

By assigning the phase shift variable φ the value $\varphi = \pi$ the time delay τ_0 gets the value $\tau_0 = 0.0020s$. This gives the ratio $\frac{\tau_0}{\tau} = \frac{0.0020s}{0.0040s} = \frac{1}{2}$, which implies stability. $\varphi = \pi$ makes it possible to write Eq. (10) as Eq. (11).

$$\begin{aligned} u(t) &= \sin(\omega t + \pi) - \sin(\omega t) \Leftrightarrow \\ u(t) &= -2\sin(\omega t) \Leftrightarrow \\ u(t) &= -2y(t) \end{aligned} \tag{11}$$

where $\omega = 2 \cdot \pi \cdot 250 \approx 1571$ rad/s. There is a transport delay d = 0.0015s, from the actuator to the sensor. This delay needs to be considered when generating the control signal. The period time $\tau = 0.0040s$ of the limit cycle makes it possible to write the control law Eq. (11) as Eq. (12)

$$\begin{aligned} u(k) &= -2y(k) \Leftrightarrow \\ u(k) &= -2y(k - (\tau - d)) \Leftrightarrow \\ u(k) &= 2y(k - (\tau - d) + \frac{\tau}{2}) \Leftrightarrow \\ u(k) &= 2y(k - 25 + 20) = 2y(k - 5) \end{aligned} \tag{12}$$

where k represents discrete time samples.

4.4 Results

A controller (App. A.2) was implemented using RTAI and the result can be viewed in Fig. 18. The result shows that the controller stabilizes the process and reduces the amplitude of the pressure. When the controller was switched on it was not longer possible to hear the sound from the Rijke tube.

4.5 Conclusions

The controller used passivity theory to reduce the energy in the system i.e. the amplitude of the pressure. Even though the controller was simple it stabilized the process. This fact indicates that a more advanced controller also will stabilize the process and probably perform even better. A Linear Quadratic Gaussian (LQG) controller that only punishes the output signal is an example of a more advanced controller.

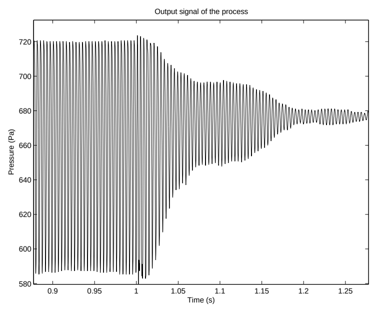

Figure 18: The controller was switched on when t = 1s.

5 Acknowledgements

This work would have never been possible without the help and support of certain people. The following words are just a mere attempt to try and express our warm thanks to them.

First of all, we would like to thank our supervisor, Dr Rolf Johansson at the Department of Automatic Control (DAC) at Lunds Institute of Technology, for introducing us to the world of thermoacoustics which has been filled with blood, sweat and tears given in exchange for the much greater joy and satisfaction felt with every progress in this thesis. Often planting ideas and suggestions in his humble way without us even noticing it until sitting in the laboratory wondering how we came up with these ideas that worked.

Fredrik Hermann, at the Department of Heat and Power Engineering at Lunds Institute of Technology, for always putting his own work aside when we needed help with setting up and modifying our "fireworks" and for giving us that little extra information regarding the mysteries of combustion's do's and don'ts. Good luck with your work!

Rolf Braun and Anders Blomdell at DAC, the spiders in the net when it comes to computers and electronics.

And finally, Ingjald Andréasson, for fixing the unfixable, finding the unfindable, cheering us up and helping us to always look at the bright side of life.

– The Authors

References

[1] Åström, J. Karl and Wittenmark, Björn(1997) *Computer Controlled Systems*, Theory and design, Prentice Hall, Upper Saddle River, NJ.

[2] J.P. Hathout, M. Fleifil, A. M. Annaswamy and A.F. Ghoniem. *Role of actuation in combustion control.* IEEE Conference on Control Applications/CASD, Hawaii, August 22-27,1999.

[3] Matveev K. *Thermoacoustic Instabilities in the Rijke Tube: Experiments and Modeling* California Institute of Technology, Pasadena, California, 2003

[4] Rijke P. L. *Annalen der Physik* 107:339, 1859

[5] Jean-Pierre Hathout. *Thermoacoustic Instability* "Reacting Gas Dynamics Computational Lab". Department of Mechanical Engineering, MIT, Cambridge, MA 02139.

[6] Slotine, Jean-Jacques E. and Li, Weiping (1991) *Applied Nonlinear Control*, Prentice Hall, Upper Saddle River, NJ

[7] Johansson, R., (1993) *System Modelling and Identification*, Prentice Hall, Englewood Cliffs

A RTAI Structures

A.1 getFrequencyScheme

A scheme over the getFrequency RTAI application can be viewed in Fig. 19.

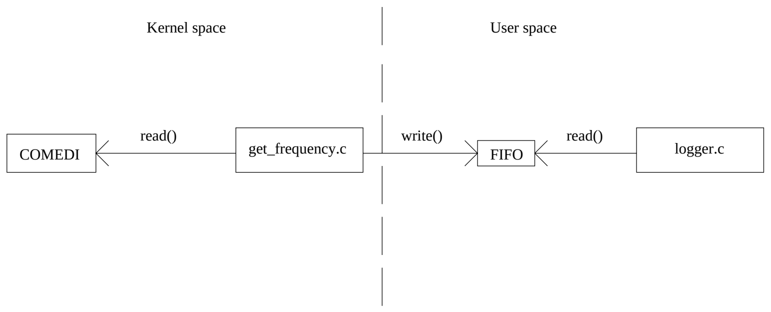

Figure 19: Scheme over the getFrequency RTAI application.

A.2 controllerScheme

A scheme over the controller RTAI application can be viewed in Fig. 20.

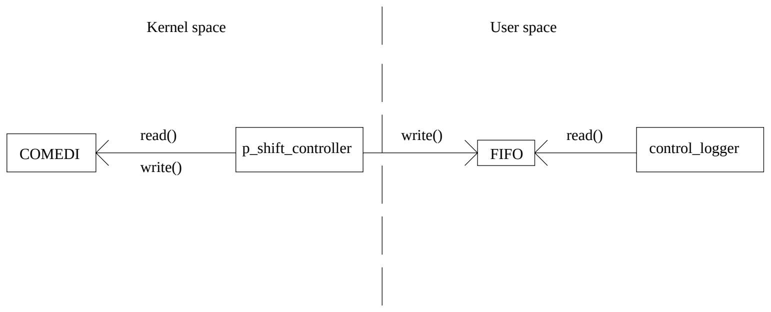

Figure 20: Scheme over the controller RTAI application.

B C-files

B.1 getfrequency.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/comedilib.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
```

```
#define SUBDEV0 0
#define ANALOGIN0 0
#define ANALOGIN1 1
#define RANGE 0
#define AREF 0
#define FIFO 0
#define H 100000 //period time (ns)
#define STACKSIZE 16536
#define PRIORITY 10

MODULE_LICENSE("GPL");

static RT_TASK sampler_task;
static comedi_t *iocard;

static void sampler(int arg)
{
  lsampl_t sample_an_in0, sample_an_in1;
  int counter = 0;
  double volts, convert_to_krange = 1000000;
  comedi_krange c_krange;
  comedi_get_krange(iocard, SUBDEV0, ANALOGIN0, RANGE,
     &c_krange);
  double convert_ratio = (c_krange.max/convert_to_krange
      - c_krange.min/convert_to_krange)/
      comedi_get_maxdata(iocard, SUBDEV0, ANALOGIN0);
  double min_tmp = c_krange.min/convert_to_krange;

  while (1) {
    comedi_data_read(iocard, SUBDEV0, ANALOGIN0, RANGE,
        AREF, &sample_an_in0);
    volts = min_tmp + convert_ratio*sample_an_in0;
    rtf_put(FIFO, &volts, sizeof(volts));
    comedi_data_read(iocard, SUBDEV0, ANALOGIN1, RANGE,
        AREF, &sample_an_in1);
    volts = min_tmp + convert_ratio*sample_an_in1;
    rtf_put(FIFO, &volts, sizeof(volts));
    counter += 1;
    if(counter == 50000){
      rt_printk("\n######## get_frequency.c is shutting
          down... ########\n");
      comedi_test_exit();
    }
    rt_task_wait_period();
  }
}

static int comedi_test_init()
{
  iocard = comedi_open("/dev/comedi0");
  if (iocard) {
    RTIME t;
    rtf_create(FIFO, 400000);
    rt_set_periodic_mode();
    rt_task_init(&sampler_task, sampler, 0, STACKSIZE,
        PRIORITY, 0, 0);
    t = start_rt_timer(nano2count(H));
    rt_task_make_periodic(&sampler_task, rt_get_time() + t, t);
  }
  return 0;
}

static void comedi_test_exit()
{
  stop_rt_timer();
  rt_task_delete(&sampler_task);
  rtf_destroy(FIFO);
  comedi_close(iocard);
}

module_init(comedi_test_init);
module_exit(comedi_test_exit);
```

B.2 logger.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <signal.h>

static int terminate;
static void terminate_process(int notUsed){ terminate = 1; }

int main(int argc, char **argv)
{
  FILE *analogin0_fd, *analogin1_fd;
  analogin0_fd = fopen("analogin0.dlm", "w");
  analogin1_fd = fopen("analogin1.dlm", "w");
  if(!analogin0_fd || !analogin1_fd){
    fprintf(stderr, "Error opening output file:\n");
    exit(1);
  }
  int fifo;
  double analogin0 = 0, analogin1 = 0;
  if((fifo = open("/dev/rtf0", O_RDONLY)) < 0){
    fprintf(stderr, "Error opening /dev/rtf0\nError code:
        %d\n", fifo);
    exit(1);
  }
  signal(SIGINT, terminate_process);
  while(!terminate){
    read(fifo, &analogin0, sizeof(analogin0));
    fprintf(analogin0_fd, " %f", analogin0);
    fprintf(analogin1_fd, " %f", analogin1);
  }
  fclose(analogin0_fd);
  fclose(analogin1_fd);
  printf("The logger terminates...\n");
  return 0;
}
```

B.3 generate_sin.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/comedilib.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
#include <math.h>
#include "constants.h"

MODULE_LICENSE("GPL");

static RT_TASK sampler_task;
static comedi_t *iocard;

static void sampler(int arg)
{
  lsampl_t out_sample;
  int counter = 0;
  double sin_value, amplitude = 1.0, freq;
  comedi_krange c_krange;
  comedi_get_krange(iocard, SUBDEV1,
    ANALOGOUT0, RANGE, &c_krange);
  double convert_from_phys_ratio =
    comedi_get_maxdata(iocard, SUBDEV1,
       ANALOGOUT0) /
    (c_krange.max/CONVERTTOKRANGE -
     c_krange.min/CONVERTTOKRANGE);

  freq = 250;   //Hz
  freq *= 2*M_PI; //convert to radians
  freq /= 1E9; //speedup
  double min_tmp = c_krange.min/CONVERTTOKRANGE;

  printk("\n\n ######## CONSTANTS #############\n");
  int tmp = min_tmp;
  printk("min_tmp: %d\n", tmp);
  tmp = c_krange.max/CONVERTTOKRANGE;
  printk("c_krange.max/CONVERTTOKRANGE: %d\n", tmp);
  tmp = c_krange.min/CONVERTTOKRANGE;
  printk("c_krange.min/CONVERTTOKRANGE: %d\n", tmp);
  tmp = comedi_get_maxdata(iocard, SUBDEV1, ANALOGOUT0);
  printk("maxdata: %d\n", tmp);
  tmp = convert_from_phys_ratio;
  printk("convert_from_phys_ratio: %d\n", tmp);
  printk("\n####################\n\n");

  while (1) {
    sin_value = amplitude *
      sin(freq*rt_get_time_ns()) + amplitude + min_tmp;
    out_sample = (sin_value -
  min_tmp)*convert_from_phys_ratio;
    int result = 100;
    int tm = sin_value*100;
    printk("Volts*100: %d\n", tm);
    comedi_data_write(iocard, SUBDEV1, ANALOGOUT0,
      RANGE, AREF, out_sample);
    counter += 1;
    rt_task_wait_period();
    //if(counter == 50000){
    //  comedi_test_exit();
    //}
  }
}

static int comedi_test_init()
{
  iocard = comedi_open("/dev/comedi0");
  if (iocard) {
    RTIME t;
    rtf_create(FIFO, 400000);
    rt_set_periodic_mode();
    rt_task_init(&sampler_task, sampler, 0,
 STACKSIZE, PRIORITY, 0, 0);
    t = start_rt_timer(nano2count(H));
    rt_task_make_periodic(&sampler_task,
  rt_get_time() + t, t);
  }
  return 0;
}

static void comedi_test_exit()
```

```
{
  stop_rt_timer();
  rt_task_delete(&sampler_task);
  rtf_destroy(FIFO);
  comedi_close(iocard);
}

module_init(comedi_test_init);
module_exit(comedi_test_exit);
```

B.4 p_shift_controller.c

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/comedilib.h>
#include <rtai.h>
#include <rtai_sched.h>
#include <rtai_fifos.h>
#include <math.h>
#include "../constants.h"

MODULE_LICENSE("GPL");

static RT_TASK sampler_task;
static comedi_t *iocard;

static void sampler(int arg)
{
  lsampl_t in_sample0, out_sample;
  int counter = 0;
  double sin_value, amplitude = 2.5, freq, volts;
  comedi_krange out_c_krange;
  comedi_get_krange(iocard, SUBDEV1,
    ANALOGOUT0, RANGE, &out_c_krange);
  double convert_from_phys_ratio = comedi_get_maxdata(iocard,
      SUBDEV1,
      ANALOGOUT0) /
    (out_c_krange.max/CONVERTTOKRANGE -
     out_c_krange.min/CONVERTTOKRANGE);

  comedi_krange in_c_krange;
  comedi_get_krange(iocard, SUBDEV0, ANALOGIN0, RANGE,
    & in_c_krange);
  double convert_to_phys_ratio =
    (in_c_krange.max/CONVERTTOKRANGE
     - in_c_krange.min/CONVERTTOKRANGE)/
    comedi_get_maxdata(iocard, SUBDEV0, ANALOGIN0);

  double out_min_tmp = out_c_krange.min/CONVERTTOKRANGE;
  double in_min_tmp = in_c_krange.min/CONVERTTOKRANGE;

  printk("\n\n ######## CONSTANTS #############\n");
  int tmp = out_min_tmp;
  printk("min_tmp: %d\n", tmp);
  tmp = out_c_krange.max/CONVERTTOKRANGE;
  printk("out_c_krange.max/CONVERTTOKRANGE: %d\n", tmp);
  tmp = out_c_krange.min/CONVERTTOKRANGE;
  printk("out_c_krange.min/CONVERTTOKRANGE: %d\n", tmp);
  tmp = comedi_get_maxdata(iocard, SUBDEV1, ANALOGOUT0);
  printk("Analog_out maxdata: %d\n", tmp);
  tmp = convert_from_phys_ratio;
  printk("convert_from_phys_ratio: %d\n", tmp);
  printk("\n####################\n\n");

  /* HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH */
  unsigned data[25] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                     0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0}; //data collection
  int current_val = 0;
  unsigned u, y;
  while (0) { //logging off/on
    counter += 1;
    u = data[current_val]; // * 1.0;
    printk(" %d\n", u);
    if(counter >= 10000){
      comedi_data_write(iocard, SUBDEV1, ANALOGOUT0,
RANGE, AREF, u);
    }
    comedi_data_read(iocard, SUBDEV0, ANALOGIN0, RANGE,
     AREF, &y);
    data[current_val] = y;
    current_val = (current_val + 1) % 25;
    if(counter == 30000){ // three seconds...quit...
      rt_printk("\n######## p_shift_controller is shutting
          down... ########\n");
      comedi_test_exit();
    }

    rtf_put(FIFO, &y, sizeof(y));
    if(counter < 10000){
      u = 0;
    }
    rtf_put(FIFO, &u, sizeof(u));

    rt_task_wait_period();
  }

  //this loop is running when the logging is swithed off...
  while (1) {
    u = data[current_val];
    printk(" %d\n", u);
    comedi_data_write(iocard, SUBDEV1, ANALOGOUT0,
      RANGE, AREF, u);
    comedi_data_read(iocard, SUBDEV0, ANALOGIN0,
     RANGE, AREF, &y);
    data[current_val] = y;
    current_val = (current_val + 1) % 25;
    rt_task_wait_period();
  }
}

static int comedi_test_init()
{
  iocard = comedi_open("/dev/comedi0");
  if (iocard) {
    RTIME t;
    rtf_create(FIFO, 200000);
    rt_set_periodic_mode();
    rt_task_init(&sampler_task, sampler, 0,
 STACKSIZE, PRIORITY, 0, 0);
    t = start_rt_timer(nano2count(H));
    rt_task_make_periodic(&sampler_task,
  rt_get_time() + t, t);
  }
  return 0;
}

static void comedi_test_exit()
{
  stop_rt_timer();
  rt_task_delete(&sampler_task);
  rtf_destroy(FIFO);
  comedi_close(iocard);
}

module_init(comedi_test_init);
module_exit(comedi_test_exit);
```

B.5 control_logger.c

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/fcntl.h>
#include <signal.h>

static int terminate;
static void terminate_process(int notUsed)
{ terminate = 1; }

int main(int argc, char **argv)
{
  FILE *analogin0_fd, *analogin1_fd;
  analogin0_fd = fopen("y_value.dlm", "w");
  analogin1_fd = fopen("u_value.dlm", "w");
  if(!analogin0_fd || !analogin1_fd){
    fprintf(stderr, "Error opening output file:\n");
    exit(1);
  }
  int fifo;
  unsigned analogin0 = 0, analogin1 = 0;
  if((fifo = open("/dev/rtf0", O_RDONLY)) < 0){
    fprintf(stderr,
    "Error opening /dev/rtf0\nError code:
        %d\n", fifo);
    exit(1);
  }
  signal(SIGINT, terminate_process);
  while(!terminate){
    read(fifo, &analogin0, sizeof(analogin0));
    fprintf(analogin0_fd, " %d", analogin0);
    read(fifo, &analogin1, sizeof(analogin1));
    fprintf(analogin1_fd, " %d", analogin1);
  }
  fclose(analogin0_fd);
  fclose(analogin1_fd);
  printf("The control_logger terminates...\n");
  return 0;
}
```

C Header Files

C.1 constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#define RANGE 0
#define AREF 0
#define FIFO 0
#define H 100000
#define STACKSIZE 16536
#define PRIORITY 10
#define CONVERTTOKRANGE 1000000
#define SUBDEV0 0
#define SUBDEV1 1
```

```
#define ANALOGIN0 0
#define ANALOGIN1 1
#define ANALOGOUT0 0

#endif
```

D Matlab Files

D.1 getFrequency.m

```
analogin0 = dlmread('analogin0_031119.dlm');
analogin1 = dlmread('analogin1_031119.dlm');
ts = 0.0001;
t = 0:ts:5;
figure(1);
hold on;
plot(t, analogin0);
plot(t, analogin1);
figure(2);
hold on;
dat1 = iddata(analogin0', t', ts);
datf1 = fft(dat1);
plot(datf1);
figure(3);
hold on;
dat2 = iddata(analogin1', t', ts);
datf2 = fft(dat2);
plot(datf2);
```

D.2 getParModel.m

```
%this m-file fetches parametric models...

% ################# ARX ####################
if(0)
  NN = struc([3], [3], [1 2 3 4 5 6 7 ...
    8 9 10 15 20 25 50])

  corrCoeffs = zeros(size(NN, 1));
  for i = 1:size(NN, 1)
    m = arx(id_dat, NN(i, :));
    yh = idsim(uv, m);
    c = corrcoef(yh, yv);
    corrCoeffs(i) = c(1, 2);
    i
  end

  V = arxstruc(zi, zv, NN)
  newNN = selstruc(V)
  m = arx(id_dat, newNN)

end

% ################ ARMAX ##################
if(1)
  aDegree = 3;
  bDegree = 3;
  cDegree = 3;
  tDelays = 2;
  corrCoeffs = zeros(aDegree*bDegree*cDegree*tDelays);
  tmpDelay = 0;
  index = 0;
  structMatrix = zeros(aDegree*bDegree*cDegree*tDelays, 4)
  for a = 1:aDegree
    for b = 1:bDegree
      for c = 1:cDegree
for d = 1:tDelays
  index = index + 1
  tmpDelay = 15;
  if d == 2
    tmpDelay = 25;
  end
  structMatrix(index, :) = [a b c tmpDelay];
  m = armax(id_dat, structMatrix(index, :));
  yh = idsim(uv, m);
  co = corrcoef(yh, yv);
  corrCoeffs(index) = co(1, 2);
end
      end
    end
  end
end

if(1)
  aDegree = 2;
  bDegree = 2;
  cDegree = 1;
  maxTDelay = 30;
  structMatrix2 = zeros(maxTDelay, 4);
  corrCoeffs2 = zeros(maxTDelay);
  for tDelay = 1:30
    tDelay
    structMatrix2(tDelay, :) = [aDegree bDegree cDegree tDelay];
    m = armax(id_dat, structMatrix2(tDelay, :));
    yh = idsim(uv, m);
    co = corrcoef(yh, yv);
    corrCoeffs2(tDelay) = co(1, 2);
  end
end
```

D.3 getNonParModel.m

```
figure
[IR, R, CL] = cra(id_dat, 100, 0, 0); % no prewhitening
plot(R(100:200, 4)); %plotting the correlation function
                     %between ui and y
title('Correlation function of the input and output signals.');
xlabel('Time delay (sample)');
ylabel('Correlation coefficient');

figure
[IR, R, CL] = cra(id_dat, 100, [], []); %default prewhitening
plot(R(100:200, 4));

figure
spectrum(ui, yi, [], [], [], fs);

std_output = std(yi)
```

D.4 validateModels.m

```
%script validateModels.m

figure(1)
yh = idsim(uv, m);
%t1 = 0:ts:0.9999;
t1 = 0:ts:4.9999;
%plot(t1, yh, 'r', t1, yv, 'b', t1, uv, 'g');
plot(t1, yh, 'r', t1, yv, 'b');
correlationM = corrcoef(yh, yv)

figure(2)
residuals = yv - yh;
[P, F] = spectrum(residuals, 256, 100, [], fs);
plot(F, P(:, 1), 'b-');
title('Power spectrum of the residuals');
xlabel('Frequency');
ylabel('Power');

figure(3)
zpDisc = th2zp(m)
zpplot(zpDisc);

figure(4)
modelC = thd2thc(m)
zpCont = th2zp(modelC)
zpplot(zpCont);

figure(5)
[H Snn] = th2ff(m);
```

```
bodeplot(H);
figure(6)
bodeplot(Snn);
```

D.5 iScript.m

```
%this m-file initiates the necessary variables
%for system identification.

ts = 0.0001;
fs = 1/ts;
%full_size = 20000;
%half_size = 10000;
full_size = 100000;
half_size = 50000;
u = u/10;
z = [y' u'];
zi = z(1:half_size, :);
zi = dtrend(zi);
yi = zi(:, 1);
ui = zi(:, 2);
id_dat = iddata(yi, ui, ts);
zv = z(half_size + 1 : full_size, :);
zv = dtrend(zv);
yv = zv(:, 1);
uv = zv(:, 2);
val_dat = iddata(yv, uv, ts);
```

E System Identification

File: i1.mat

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 250 Hz and 260 Hz.
Amplitude = 1 V
Phase = 0 degrees

File: i11.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 250 Hz and 260 Hz.
Amplitude = 1 V
Phase = 0 degrees

File: i2.mat

Information:

100 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 250 Hz and 260 Hz.
Amplitude = 2.5 V
Phase = 0 degrees

File: i21.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 250 Hz and 260 Hz.
Amplitude = 2.5 V
Phase = 0 degrees

File: i3.mat

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 250 Hz and 260 Hz.
Amplitude = 0.5 V
Phase = 0 degrees

File: i31.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 250 Hz and 260 Hz.
Amplitude = 0.5 V
Phase = 0 degrees

File: i4.mat

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the amplitudes 1 V and 1.5 V.
Frequency = 250 Hz
Phase = 0 degrees

File: i41.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the amplitudes 1 V and 1.5 V.
Frequency = 250 Hz
Phase = 0 degrees

File: i5.mat

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the amplitudes 1 V and 2.5 V.

Frequency = 250 Hz
Phase = 0 degrees

File: i51.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the amplitudes 1 V and 2.5 V.
Frequency = 250 Hz
Phase = 0 degrees

File: i6.mat

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the phases 0 and 100 degrees
Frequency = 250 Hz
Amplitude = 1 V

File: i61.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the phases 0 and 100 degrees
Frequency = 250 Hz
Amplitude = 1 V

File: i7.mat

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the phases 0 and 100 degrees
Frequency = 250 Hz
Amplitude = 2.5 V

File: i71.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the phases 0 and 100 degrees
Frequency = 250 Hz
Amplitude = 2.5 V

File: i9.mat

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 220 Hz and 225 Hz.
Amplitude = 2.5 V
Phase = 0 degrees

File: i91.mat

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 220 Hz and 225 Hz.
Amplitude = 2.5 V
Phase = 0 degrees

File: $i10_{.}mat$

Information:
100 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 220 Hz and 225 Hz.
PRBS on the phases 0 and 10 degrees.
PRBS on the amplitudes 2 V and 2.5 V.

File: $i10_1.mat$

Information:
1000 sampling intervals between PRBS shift register updates.
PRBS on the frequencies 220 Hz and 225 Hz.
PRBS on the phases 0 and 10 degrees.
PRBS on the amplitudes 2 V and 2.5 V.

File: $i11_{.}mat$

Information:
No PRBS signal. The input signal contains the frequencies 210, 220, 225, 230, 240, 300 Hz A = 2.5 V Phase = 0 degrees

F Makefile

F.1 Makefile

```
INCLUDE_RTAI=/usr/src/rtai.i386/include
INCLUDE_LINUX=/usr/src/linux-2.4.20-up/include
RTAI=/lib/modules/2.4.20-rthal5-up/rtai/
SYSMODULES=/lib/modules/2.4.20-rthal5-up/rtai

CC=gcc
CFLAGS= -I./include \
-Wall \
-O2 -g
MODULE_CFLAGS= -I./include \
-I$(INCLUDE_RTAI) \
-I$(INCLUDE_LINUX) \
-march=i586 \
-include /usr/src/linux-2.4.20-up/include/linux/modversions.h \
-Wall \
-O2 \
-D__KERNEL__ -DMODULE -DMODVERSIONS

SFLAGS= -I./include \
-I$(INCLUDE_RTAI) \
-Wall \
-O2 -g

all:  compiled/i386/logger \
compiled/i386/get_frequency.o \
compiled/i386/generate_sin.o \
compiled/i386/ex.o \
compiled/i386/collect_data.o \
compiled/i386/receive_data \
compiled/i386/logger_2 \
compiled/i386/flame_holder_test.o \
compiled/i386/p_shift_controller.o \
compiled/i386/control_logger

compiled/i386/%: %.c
$(CC) $(CFLAGS) -o $@ $*.c

compiled/i386/%.o: %.c
$(CC) $(MODULE_CFLAGS) -o $@ -c $*.c

compiled/i386/%.o: driver/%.c
$(CC) $(MODULE_CFLAGS) -o $@ -c driver/$*.c

compiled/i386/get_frequency.o: \
experiments/get_frequency.c constants.h
$(CC) $(MODULE_CFLAGS) -o $@ -c $<

compiled/i386/logger: experiments/logger.c
$(CC) $(SFLAGS) -o $@ $<

compiled/i386/generate_sin.o: generate_sin.c constants.h
$(CC) -nostdlib -Wl,-r $(MODULE_CFLAGS) -o $@ $< -lm

compiled/i386/ex.o: ex.c constants.h
$(CC) -nostdlib -Wl,-r $(MODULE_CFLAGS) -o $@ $< -lm

compiled/i386/collect_data.o: \
identification/collect_data.c constants.h
$(CC) -nostdlib -Wl,-r $(MODULE_CFLAGS) -o $@ $< -lm

compiled/i386/receive_data: identification/receive_data.c
$(CC) $(SFLAGS) -o $@ $<

compiled/i386/flame_holder_test.o: \
experiments/flame_holder_test.c constants.h
$(CC) -nostdlib -Wl,-r $(MODULE_CFLAGS) -o $@ $< -lm

compiled/i386/logger_2: experiments/logger_2.c
$(CC) $(SFLAGS) -o $@ $<

compiled/i386/p_shift_controller.o: \
control/p_shift_controller.c constants.h
$(CC) -nostdlib -Wl,-r $(MODULE_CFLAGS) -o $@ $< -lm

compiled/i386/control_logger: control/control_logger.c
$(CC) $(SFLAGS) -o $@ $<
```

G Run Scripts

G.1 startup

This script mounts RTAI.

```
#!/bin/sh
insmod rtai
insmod rtai_sched_up
insmod rtai_fifos
insmod comedi
insmod amcc_s5933
insmod adv_pci1710
insmod kcomedilib
/usr/sbin/comedi_config /dev/comedi0 pci1711
```

G.2 removeM

This script unmounts RTAI.

```
rmmod kcomedilib
rmmod adv_pci1710
rmmod amcc_s5933
rmmod comedi
rmmod rtai_fifos
rmmod rtai_sched_up
rmmod rtai
```

G.3 run

This script mounts RTAI, starts an application and unmounts RTAI, when the application quits.

```
#!/bin/sh

####### Default mounting... ########
insmod rtai
insmod rtai_sched_up
insmod rtai_fifos
insmod comedi
insmod amcc_s5933
insmod adv_pci1710
insmod kcomedilib
/usr/sbin/comedi_config /dev/comedi0 pci1711
####################################

######## controller ################
insmod compiled/i386/p_shift_controller.o
./compiled/i386/control_logger
rmmod p_shift_controller

######## flame_holder_test #########
#insmod compiled/i386/flame_holder_test.o
#./compiled/i386/logger_2
#rmmod flame_holder_test
####################################

######## get_frequency #############
#insmod compiled/i386/get_frequency.o
#./compiled/i386/logger
#rmmod get_frequency
####################################

######## PRBS generating ###########
#insmod compiled/i386/collect_data.o
#./compiled/i386/receive_data
#rmmod collect_data
####################################

###### Default unmounting... #####
rmmod kcomedilib
rmmod adv_pci1710
rmmod amcc_s5933
rmmod comedi
rmmod rtai_fifos
rmmod rtai_sched_up
rmmod rtai
##################################
```

H Amplifier and Loudspeaker

The loudspeaker has the following characteristics:

Type: PC77DU60-02FP
Nominal impedans: 8 Ω
Maximum power: 5 W
Frequency range: 160-18000 Hz

Resonant frequency: 160 Hz
Height: 39 mm
Width: 78 mm

To determine the maximum voltage that could be coupled to the loudspeaker the following equation was used: $\hat{u} = \sqrt{2} \cdot \sqrt{P \cdot R} = \sqrt{2} \cdot \sqrt{5 \cdot 8} = 8.9443V$, where $\hat{u}$, P and R are the peak to peak value of the voltage, the power and the impedans respectivily.

The AD/DA card can generate an analog output voltage of 0-5 V. The power generated from the card is not enough to drive the loudspeaker. A KEMO 88032 12 W universal amplifier was coupled between the analog output and the loudspeaker. In order to control the magnitude of the output signal generated by the amplifier a 0.5 k potentiometer and a resistance was coupled between the input signal and the amplifier (see H. A power main supply was coupled to the amplifier.

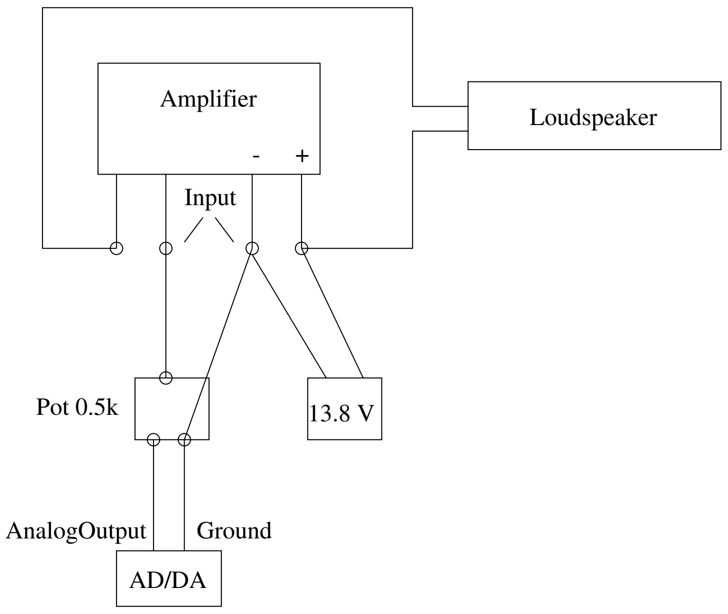

Figure 21: The amplifier and loudspeaker.