ISSN 0280-5316
ISRN LUTFD2/TFRT--5711--SE

Combustion control of the Homogenous Charge Compression Ignition dynamics

Roland Pfeiffer

Department of Automatic Control
Lund Institute of Technology
September 2003

Department of Automatic Control
Lund Institute of Technology
Box 118
SE-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
September

Document Number
ISRN LUTFD2/TFRT--5711--SE

Author(s)
Roland Pfeiffer

Supervisor
Rolf Johansson LTH, the Department of Automatic Control.
Per Tunestål LTH, Division of Combustion Engines

Sponsoring organization

Title and subtitle
Combustion control of the Homogenous Charge Compression Ignition dynamics (Förbränningsreglering av HCCI-dynamik)

Abstract

The HCCI engine has potential to replace the spark ignition and compression ignition engines of today. One of the main problems in making the engine commercially attractive is that there are no direct means of controlling the ignition phasing. This thesis attempts to describe a method for system identification of the HCCI process, and development of an effective LQG regulator for the combustion process. Matlab and Simulink are used in computations and simulations.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
59

Recipient's notes

Security classification

Master thesis

Combustion control of the Homogenous Compression Charge Ignition dynamics

System identification and development of an LQG *controller for the ignition phasing*

Performed at Lund Institute of Technology
at the
Department of Automatic Control
in association with the
Division of Combustion Engines

Performed by:
Roland Pfeiffer
Supervisors:
Professor Rolf Johansson
Ass. Professor Per Tunestål

1 Introduction

Combustion engines are very important to every one of us. They are used to power vehicles as well as electrical power generators, mobile pumps and so on. However current combustion engines all have some drawback. Spark Ignition (SI) engines have low emissions but high fuel consumption. Compression Ignition (CI) engines have low fuel consumption but high emissions. An attempt to obtain the best behaviour from both SI and CI engines is the Homogeneous Charge Compression Ignition (HCCI) engine. It has the low fuel consumption of the CI engine combined with the low emissions of the SI engine.

The HCCI engine does have its drawbacks however. Due to its operation there is no direct way of controlling the ignition phasing. This means that it is very hard to control the engine. As a step towards building a better controller than is available today, this master thesis aims at identifying the process and subsequently developing an effective controller. When a good model of the engine process is obtained this model can be transferred to a Matlab Simulink block. This offers the possibility to develop and test control strategies without having to test them on the real engine right away.

1.1 Method

This Thesis is based on experimental work performed on an engine located at Lund Institute of Technology (LTH). The particular engine on which this master thesis is based is a five-cylinder prototype built by SAAB and equipped with a system for variable compression ratio (CR) and variable inlet air temperature (IAT). The engine is described in more detail in section 2.1.

Matlab is used to make the computations necessary to create a Linear Quadratic Gaussian (LQG) controller. The functions used can be studied in the Matlab files listed in App. B.

1.2 Objective

The goal of this master thesis is to develop a model of how the ignition phasing of the HCCI engine is affected by the different input signals. This knowledge can be used to build simulation blocks for Matlab Simulink, which can be used for advanced simulations.

Another objective is to create a controller that is more effective than the gain scheduled PID-controllers used today. For more information on gain scheduling see (Åström and Wittenmark, 1995).

1.3 Outline of the text

The content of this text is outlined in the same order the different parts have been performed. It starts by describing how the system identification is performed. This part includes results and validation of the obtained model.

The next part of the text describes the steps taken to develop an LQG controller. This section includes development of a controller for an engine with cylinder individual IAT. A suggestion of how the controller can be implemented in Java is then given.

The next section sums up the most important points made. This section is followed by a discussion of problems and possible improvements.

After the describing text, there are two Appendices containing listings of examined operating points and Matlab m-files.

2 The HCCI process

When looking at the HCCI process the ignition phasing is the output signal that we want to control. This is affected by CR, IAT, engine speed and injected fuel amount. The HCCI process can thus be considered to be a multi input, single output system. The model can be expanded with more inputs as well as outputs. This work however targets the signals mentioned above.

The point of ignition is defined as the crank angle degree where 50% of the fuel has been

consumed (CA50). The crank angle degree is measured from top dead centre (TDC), which is located at 0°. Top dead centre is defined as the crank angle when the piston is at its highest position after compression. Another acronym that will be used in this thesis is ATDC, which stands for After Top Dead Centre.

In an HCCI engine the fuel is injected outside the cylinder into the inlet manifold, alternatively if direct injection is used, the fuel is injected early enough so that the fuel and air will have time to form a homogeneous mixture. This is the same procedure as in an ordinary SI engine. The advantage of this injection strategy is that the air/fuel mixture is homogeneous when combustion occurs. This leads to cleaner exhausts than would be the case with direct injection as used in CI engines. The air/fuel mixture then auto ignites from the temperature rise that occurs as a result of the piston compressing the mixture.

2.1 The engine

The engine used for the experiments performed as a part of this master thesis has five cylinders and a displacement volume of 1.6 litres. It is built by SAAB and originally built as an SI engine with variable CR. The engine has subsequently been converted to HCCI operation at LTH, and equipped with a heat exchange system which allows the heat in the exhaust gas to be used to heat the inlet air. A valve is then used to select how much of the inlet air should be taken from the heat-exchanging device or from unheated air.

Table 1 Engine specifications

Displacement	1598 cm^3 (320 cm^3/cyl)
Nr. of cylinders	5
Compression ratio	Adjustable 9 - 21:1
Bore x Stroke	68mm x 88mm

A problem is that the variable CR and IAT affects all cylinders in parallel and thus give no way to affect the ignition phasing for each cylinder individually. To achieve balanced ignition phasing for all cylinders a system that affects the cylinder individual load is used. The engine and the load balancing system are described in more detail in (Haraldsson, 2003).

The system using cylinder individual load to achieve balanced ignition phasing is quite unfavourable since it makes it impossible to obtain maximum load from all five cylinders simultaneously. During the time this thesis is written the engine is being rebuilt so that it will be possible to use cylinder individual IAT to balance the ignition phasing. This strategy allows the individual cylinders to work at the same load.

In these tests the engine is naturally aspirated, i.e. the engine is not supercharged in any way. The fuel used in these tests is 92-octane gasoline.

2.2 Measuring IAT

The first sets of experiments revealed that the thermal element was too slow. This meant that a change in IAT would be visible in the output signal several engine cycles before it was visible in the measured input signal. As of the second set of main experiments the thermal element has been switched to a faster one.

3 Experiments

When performing system identification a disturbance is added to the input signals of the process and the output is then observed. The real process (in this case the engine) will always be subject to unwanted disturbances such as noise. This means that the disturbances that have been introduced on purpose should be as large as possible to be possible to distinguish from process noise. The real engine, like most real processes, is a non linear process. This means that it is necessary to find linearity regions where it is possible to fit a linear model to the real process. This means that it is not possible to use arbitrarily large disturbances since they would push the process out of the linearity region. A thorough description of the science

of system identification can be found in (Johansson, 1993) and (Ljung, 1993).

The first step is to obtain information about linearity regions; this information is used to decide how the main experiments should be performed. The next step is to perform the main experiments. However; analysis of the main experiments may reveal that some parts of the process have been overlooked. This in turn leads to a need to revise the test cycle and possibly to perform new initial experiments.

3.1 Process stability

The HCCI process is not completely stable over the entire operating range. This means that some kind of closed loop control of the process is necessary to keep the process in a certain range. This kind of closed loop operation can cause problems when performing system identification since it will introduce non-causal behaviour. However this non-causal behaviour will be a problem mainly when performing continuous time identification. When performing discrete time identification the non-causal behaviour will not be problem. In this thesis only discrete time identification is performed.

The process is stabilized using a weak integral control of the CR at the operating points where the process is unstable.

3.2 Initial experiments

The initial experiments are done by feeding step changes to the process inputs and observing the output. The magnitudes of the step changes are increased until non linear behaviour is observed, or until the input or output are close to going outside of normal operating range. The step response experiments are performed on each of the input signals, one at a time. The input signals are:

- Compression ratio
- Inlet air temperature
- Fuel amount
- Engine speed

Other input signals exists, for instance the fuel composition (Olsson, 2002) can be used to control the ignition phasing but this thesis focuses on the input signals listed above. Plots of the initial experiments can be viewed in Figure 1 to Figure 4.

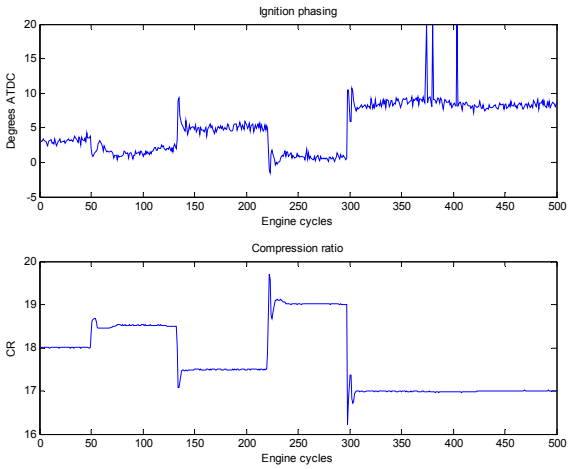

Figure 1 Step disturbances applied to the compression ratio.

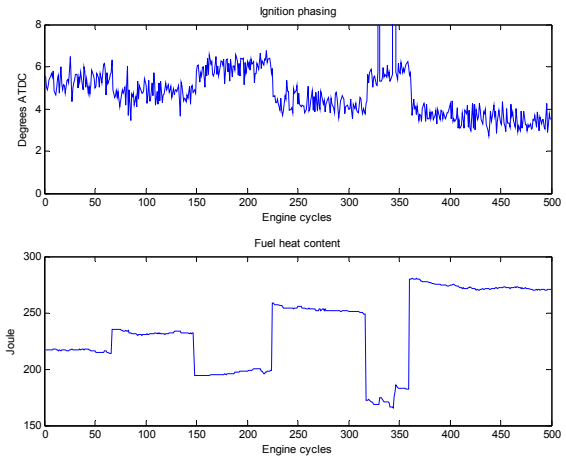

Figure 2 Step disturbances applied to the engine load.

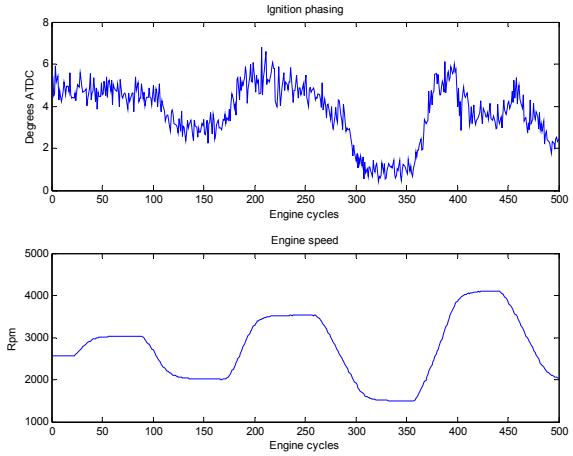

Figure 3 Step disturbances applied to the engine speed.

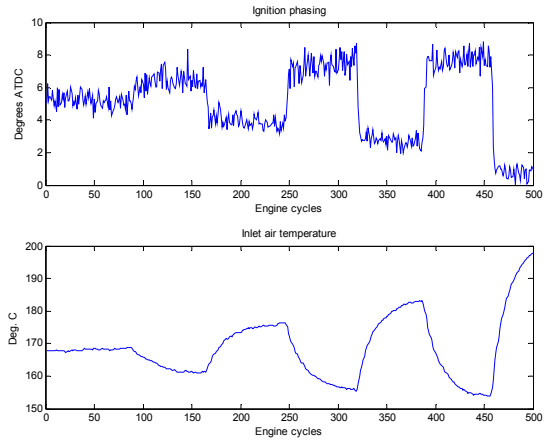

Figure 4 Step disturbance applied to the inlet air temperature. This Figure is obtained using the faster thermal element.

4 Main experiments

Test series of 1000 engine cycles provide an adequate amount of data from which the identification can be made. The operating points referred to in this section are listed in App. A.

To create the disturbances needed to perform identification of the process, Pseudo Random Binary Sequence (PRBS) signals are used. The initial experiments indicates that the process could be excited using a PRBS amplitude of ±40 J/cycle on the fuel heat (this is the heat content of the injected fuel), ±0.3 units on compression ratio, ±300 rpm on engine speed and ±0.5 % units on the valve regulating the inlet air temperature.

When performing the tests, excitation is applied to all four input signals concurrently. This means that if all disturbances work in the same direction the process will be pushed out of the linear range even though the individual input signals are in range. This was found to be a problem when performing the first round of main experiments. To minimize the risk of this the concurrent excitation levels are chosen to be 50% of the values stated above.

Four sets of experiments were conducted. In the first set the main focus was on step response analysis. At this point it was discovered that the thermal element measuring the IAT was too slow to be of any use in the subsequent tests, thus the position of the valve controlling the air flow was focused upon instead. This signal is called PWM and is a value between 0 and 1 where 0 means; valve completely closed; only cold air used, and 1 means valve completely open; only hot air used. In the very first tests performed the PWM signal was not stored.

The PWM signal is really a measure of the valve position. This signal is then transferred to the valve as a Pulse Width Modulated (PWM) signal.

The second set of experiments aimed mainly at investigating different amplitudes and shortest period of the PRBS signals. It was discovered at this point that a slower PRBS excitation on the fuel amount allowed for better results when performing system identification. A slower PRBS means that the shortest time a value is held is increased.

When performing the third set of experiments the thermal element was changed to a much faster one. This new element allows the focus to be shifted from the PWM to the IAT instead. This is of course much more interesting since the results would be valid even though the valve might be replaced. At this time the possibility to add PRBS disturbances to the engine speed has also been added.

The fourth set is largely the same as set number three, and was performed mainly to supply validation data.

Plots of the input and output signals obtained from the experiment indexed 17 can be viewed in Figure 5 to Figure 9. Only values from engine cycle 200 up to 300 are included in the plots. More values would make the plots hard to read.

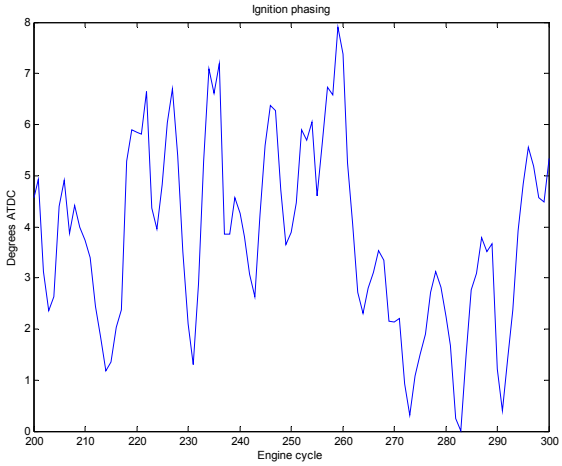

Figure 5 Measured ignition phasing for the experiment indexed 17.

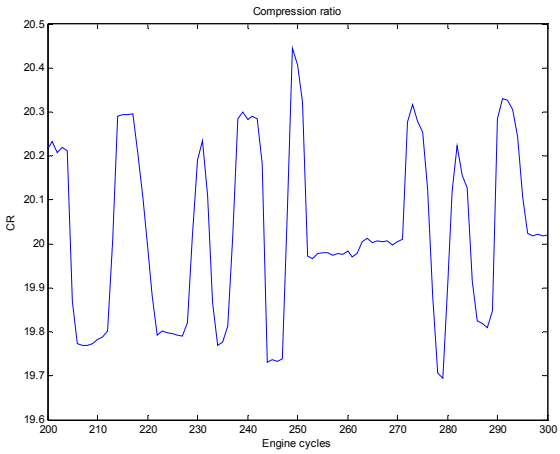

Figure 6 Compression ratio for the experiment indexed 17.

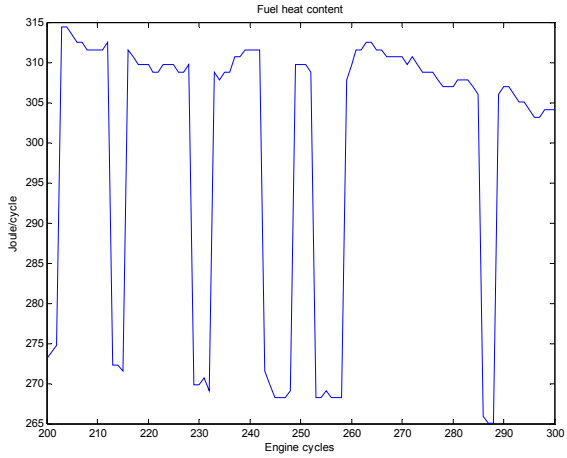

Figure 7 Engine load for the experiment indexed 17.

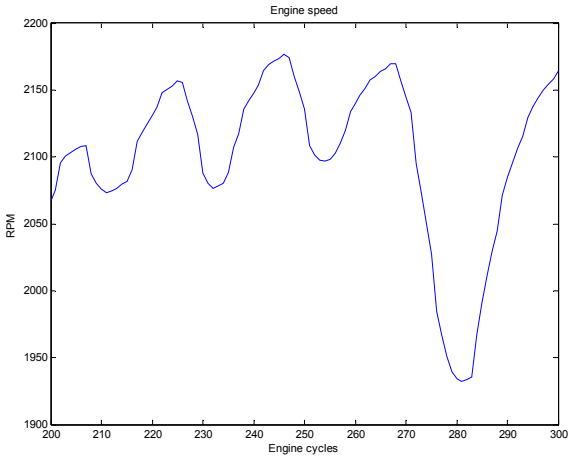

Figure 8 Engine speed used at experiment indexed 17.

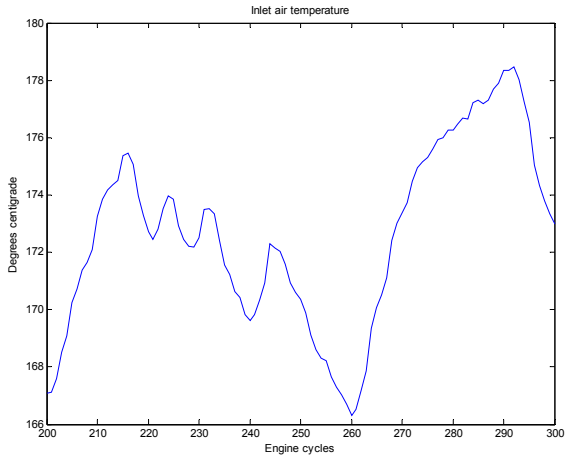

Figure 9 Inlet air temperature used at experiment indexed 17.

5 System identification

To be able to perform system identification it must be possible to distinguish the disturbances in the output signal caused by the excitation from the disturbances caused by inherent noise. One way to measure the amount of excitation of the output signal is to calculate the signal to noise ratio. This can be calculated using Eq. (1). In this Equation std(CA50) denotes the standard deviation of the output signal.

$$S/N = \frac{std(CA50)_{Excited}}{std(CA50)_{Unexcited}} \quad (1)$$

It is found that the signal to noise ratio is approximately 2.6. This should be adequate to succeed when performing identification.

5.1 Method

The Matlab function n4sid is used to perform system identification on the data collected from the experiments. This function makes use of a subspace algorithm described in (Ljung 1999).

5.2 Model accuracy

This text contains numerous mentions of correlation between simulated and measured output. Whenever this is mentioned the measured data will not be the same as the data used to create the model. The reason for this is of course that it is easy to create a perfect model by means of interpolation if the same data is used for identification as well as validation. The "perfect" model would however perform very poorly when compared to another measurement.

Correlation between simulated and measured output is a good measure of how well the model manages to mimic the behaviour of the real engine. The correlation is a value that will be at most 1. 1 means that the model is a perfect copy of the real process.

5.3 Sampling rate

The time base for the combustion process is not seconds, but engine cycles. The combustion event takes place every second revolution of the crankshaft, i.e. every engine cycle. The setting of the fuel injection, and the output, i.e. CA50, are both discrete signals, with one value for every engine cycle. IAT, CR and engine speed are continuous signals but only affect the process once every cycle. Therefore, it makes sense to base the identification, and subsequently the controller, on the time base of one engine cycle. As a consequence of this the sampling rate is limited to 1 sample per cycle. This approach avoids some of the problems encountered in earlier approaches to characterize the dynamic behaviour of the reciprocating combustion engine, see (Welbourn et al. 1959) or (Bowns 1971).

5.4 Result

Previous work (Olsson et al., To be published 2004) has shown significant cylinder-to-cylinder variation on an HCCI engine operating on two different fuels. This phenomenon was expected to be found in this engine as well. However the analysis shows that a model created by using data from all five cylinders performed equally well as a model created using only data from the examined cylinder. Thus it is concluded that there is little cylinder-to-cylinder variations in the engine on which this thesis is based.

Identification of a third order model from file with id. 18 gives the correlations listed in Table 2. In this table Corr. 1 indicates correlations between measured output and output produced by a model based only on data from the same cylinder as the validation data. Corr. 2 indicates correlations between measured output and output created using a model based on data from all five cylinders. As can be seen the difference between the models is small.

Table 2

Cylinder	1	2	3	4	5
Corr. 1	0.91	0.89	0.89	0.87	0.91
Corr. 2	0.90	0.89	0.89	0.88	0.90

The small cylinder-to-cylinder variations refer to the dynamics of the different

cylinders. Even though the different cylinders react similarly to an increase in IAT they will produce different outputs at steady state since there are different heat losses for the inlet air for the different cylinders.

5.5 Choice of model complexity

It is often the case that a large order model will perform better than a small order model. However this does not mean that the larger order model is necessarily the better choice. In general it is better to choose a smaller order model as long as it performs almost as well as the larger order models. The smaller order model will have fewer parameters that have to be determined. The parameters can thus be determined more accurately.

5.5.1 Method

The order of the model is decided by comparing the accuracy of models of several orders. This test is performed using the measurements of data set three. For each model order between 1 and 15 a model is constructed using data from each of the measurements of the set. The correlation between measured and simulated process output is then calculated for each of the different operating points of the set. The minimum, maximum and mean correlation is then plotted in Figure 10. The Matlab script for these operations can be viewed in App. B.3.

5.5.2 Result

As can be seen in Figure 10, the accuracy seems to increase up to order 3. After that the accuracy is more or less constant up to order 12 where it deteriorates slightly. Since a low order model is to prefer to a high order model as long as the quality of the predicted output is in the same region, it is decided to use a third order model. The Matlab script used to create the plot can be viewed in App. B.3. The script produces two plots, they are similar that is why only one is shown here.

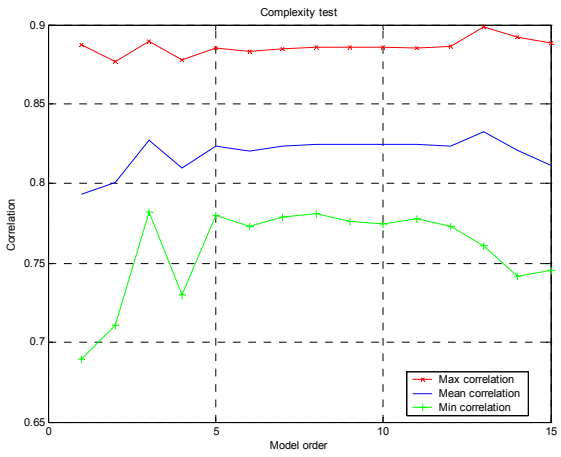

Figure 10 Correlations between measured and simulated process output for model orders between 1 and 15. The lines between the different model orders have the soul purpose of making the plot more readable.

5.6 One model or several?

It is a well-known fact that the engine changes behaviour from one operating point to another. The question that has to be answered is whether or not the process varies enough to demand several models to simulate its behaviour.

5.6.1 Method

One (third order) model for each operating point in experiment set three and four is created. The correlation between the models and validation data from that same operating point is then calculated. A model created using data from all operating points in set three and four is also created, this model will henceforth be called the merged model. The correlation between the merged model and validation data from each operating point is then calculated and compared to the value obtained using the operating point individual models.

5.6.2 Result

As can be expected the operating point individual models always produces a better result than the merged model. However, the merged model is at all operating points performing approximately equally well as the individual models. A graph showing the

result is found in Figure 11. The correlation even for the merged model is never below 0.76. A correlation of 0.76 is a very good value indeed. It is thus decided that one model is sufficient to describe the process at all the examined operating points. The graph was produced using a Matlab script which can be viewed in App. B.4.

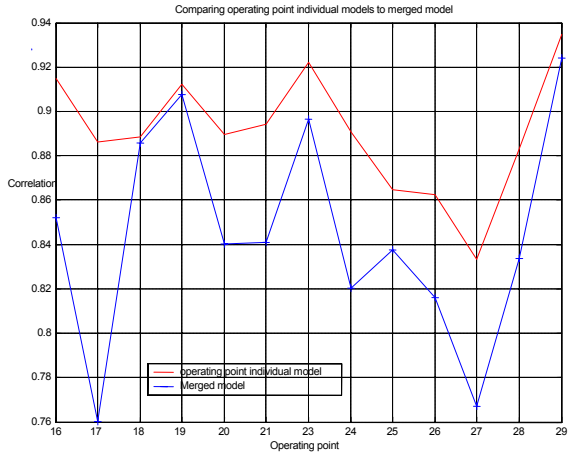

Figure 11 Correlation between measured and simulated output for operating point individual models compared to a merged model. The lines between the operating points have the soul meaning of making the plot more readable.

5.7 The model

From the results obtained above it is reasonable to deduce that a single third order model is capable of simulating the process adequately over the range of tested operating points.

5.7.1 Method

The model is created using data from all cylinders in experiments indexed 15-21 and 23-29. The Matlab code for creation of the model can be found in App. B.6.

5.7.2 Result

The model is a third order discrete state space model with the following structure:

$$\begin{aligned} x(t+1) &= Ax(t) + Bu(t) + w(t) \\ y(t) &= Cx(t) + Du(t) + e(t) \end{aligned} \qquad (2)$$

In Eq. (2), *w(t)* and *e(t)* are white process and measurement noise respectively. The variance of *w(t)* and *e(t)* is investigated in section 6.4. *y(t)* is CA50 and *u(t)* is a vector consisting of CR, IAT, fuel heat and engine speed.

The matrices in (2) are as follows:

$$A = \begin{pmatrix} 0.46236 & 0.57407 & -0.44665 \\ -0.027976 & 0.62385 & 0.66933 \\ 0.19146 & 0.055278 & 0.52031 \end{pmatrix}$$

$$B = \begin{pmatrix} 6.699\cdot10^{-3} & 2.381\cdot10^{-3} & -9.2023\cdot10^{-6} & -6.446\cdot10^{-6} \\ 2.607\cdot10^{-3} & 1.256\cdot10^{-3} & -3.631\cdot10^{-5} & 5.803\cdot10^{-6} \\ -2.746\cdot10^{-3} & -1.875\cdot10^{-3} & 2.514\cdot10^{-5} & -3.618\cdot10^{-6} \end{pmatrix}$$

$$C = \begin{pmatrix} 225.86 & 77.984 & 31.742 \end{pmatrix}$$

$$D = \begin{pmatrix} -9.3840 & -1.4295 & -6.2237\cdot10^{-4} & 2.7276\cdot10^{-3} \end{pmatrix}$$

5.7.3 Poles and zeros

The poles and zeros of the model are displayed in Figure 12 to Figure 15. Since there are four input signals there consequently are four pole-zero maps. The poles are the same for all maps, only the zeros change.

As can be seen in the Figures the model is stable. However there is a pole close to 1 on the real axis. The real process is known to be unstable (Since the value of CA50 will drift at some operating points even though the input signals are held constant). It is then likely that this pole travels over to the right of 1 at these operating points.

The zeros from CR, IAT and speed are all stable. But as with the poles there is one close to the unit circle for CR and two for IAT. From fuel heat to ignition phasing there are a couple of unstable resonant zeros. These might cause problems when it comes to controlling the process.

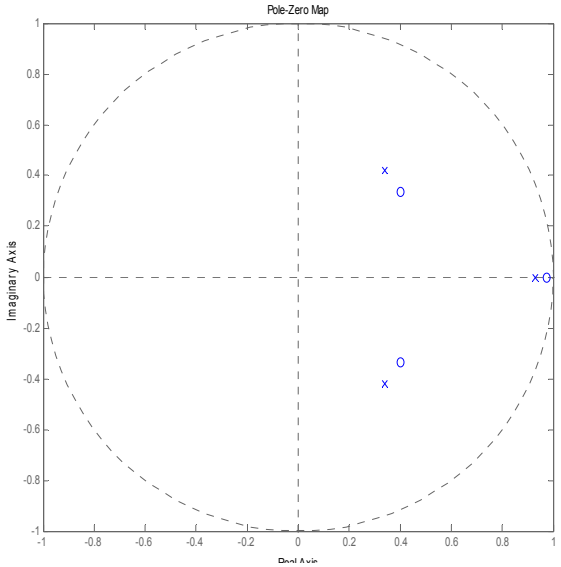

Figure 12 Pole-zero map from compression ratio to ignition phasing (CA50). Poles are denoted x and zeros are denoted o.

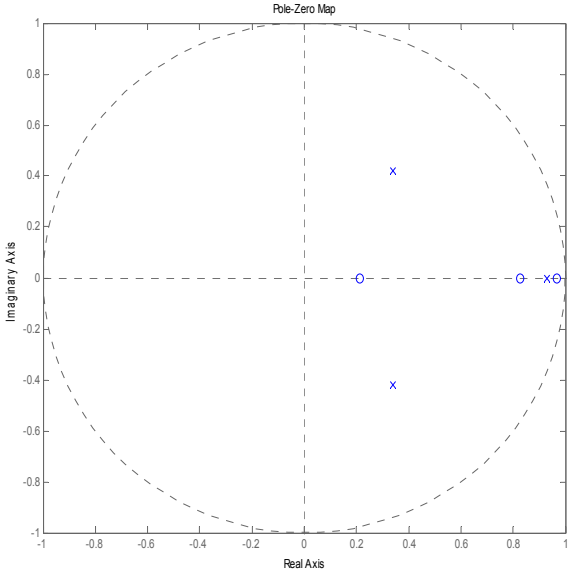

Figure 13 Pole-zero map from inlet air temperature to ignition phasing (CA50). Poles are denoted x and zeros are denoted o.

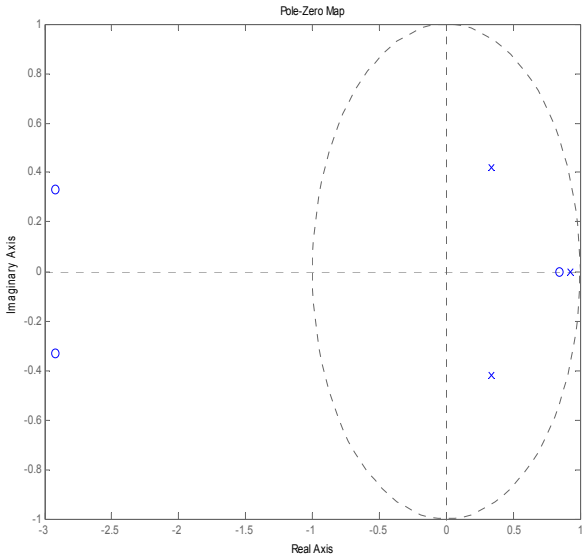

Figure 14 Pole-zero map from fuel heat to ignition phasing (CA50). Poles are denoted x and zeros are denoted o.

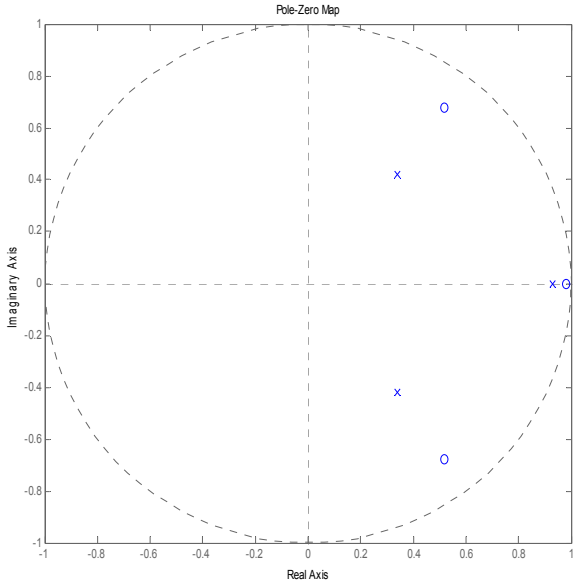

Figure 15 Pole-zero map from engine speed to ignition phasing (CA50). Poles are denoted x and zeros are denoted o.

5.7.4 Bode diagrams

The Bode diagram from each of the four studied inputs to ignition phasing can be studied in Figure 16 to Figure 19.

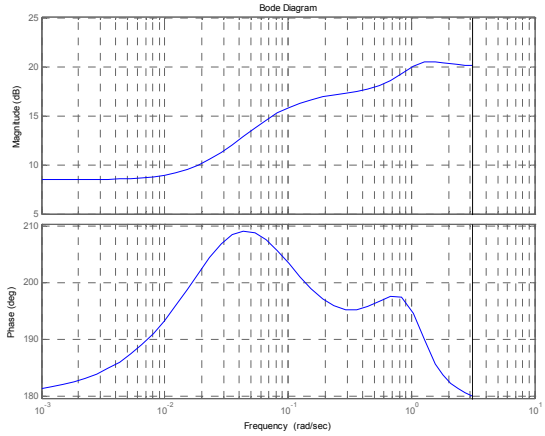

Figure 16 Bode plot from compression ratio (CR) to ignition phasing (CA50)

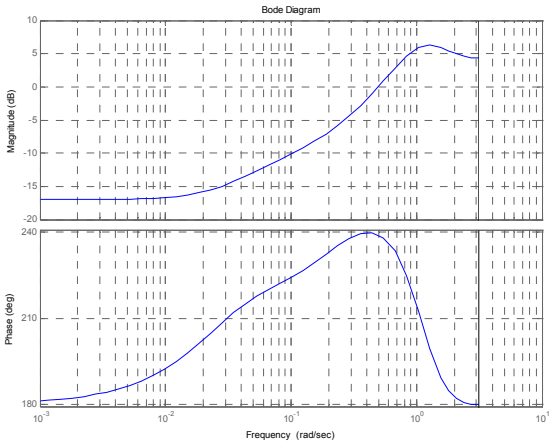

Figure 17 Bode plot from inlet air temperature (IAT) to ignition phasing (CA50)

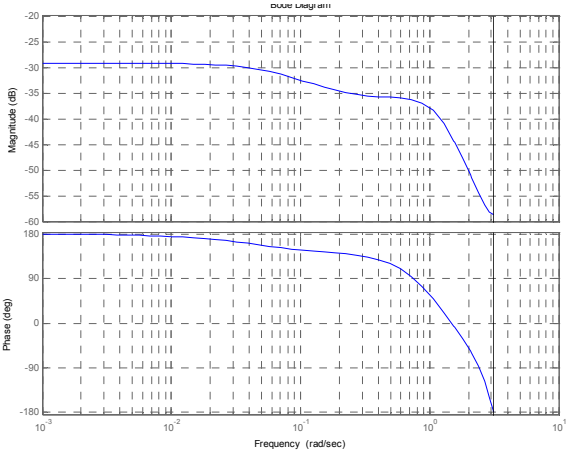

Figure 18 Bode plot from fuel heat to ignition phasing (CA50)

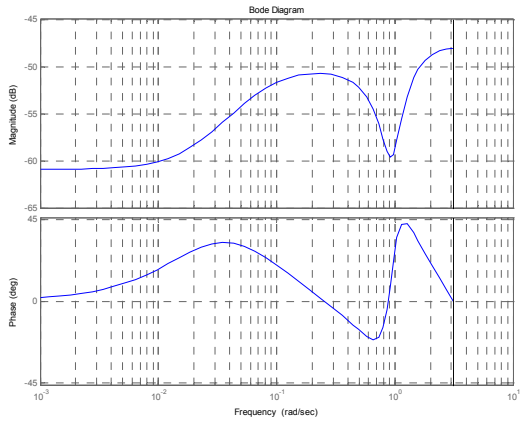

Figure 19 Bode plot from engine speed to ignition phasing (CA50)

5.7.5 Observability and controllability

The observability and controllability of the obtained model is examined using Matlab script observability_controllability_test.m which can be viewed in App. B.7.

$$W_o = \begin{pmatrix} C \\ CA \\ CA^2 \end{pmatrix} \tag{3}$$

The observability matrix W_o obtained via (3) is

$$W_o = \begin{pmatrix} 225.9 & 78.0 & 31.7 \\ 108.3 & 180.1 & -32.2 \\ 38.9 & 172.7 & 55.4 \end{pmatrix}$$

which has full rank. This means that the model is observable.

$$W_c = \begin{pmatrix} B & AB & A^2B \end{pmatrix} \quad (4)$$

The controllability matrix W_c given by (4) is a 12 by 3 matrix. Because of its large size it is not included here. It does however have rank 3, i.e. the model is controllable as well.

If the model of the process is correct this means that the real process is observable as well as controllable. These are prerequisites when the process is to be controlled.

5.7.6 Simulation capacity

The most interesting property of the model is its capacity to mimic the behaviour of the real process, i.e. the engine. The final model is the model used to create Figure 11. As can be seen in this Figure; the lowest correlation of all operating points is obtained when the model is compared to the file indexed 17. Since this operating point is the one with the lowest correlation this is the operating point used to create Figure 20. The results from the other examined operating points are equally good or better.

Even though this is the worst-case scenario (at least among the operating points that have been tested) the result is quite good.

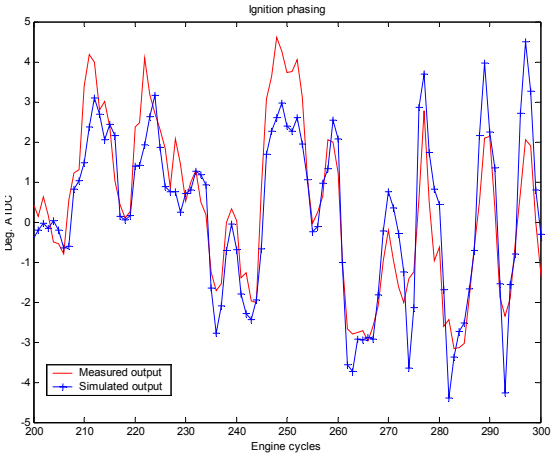

Figure 20 Simulated and measured output. Data used is from file indexed 17.

5.8 Validation

With a perfect model the remaining residuals, i.e. the part of the simulated output signal that does not match the measured output signal, is white noise[1]. All physical processes contain noise of some form.

Most plots in this section are created using data from file indexed 17. This is done since this is the operating point with the lowest correlation between measured output and output simulated using the obtained model. All other operating points can thus be expected to produce results that are equally good or better.

5.8.1 Residuals

The frequency power spectrum of the residuals when comparing measured and simulated output at operating point indexed 17 can be viewed in Figure 21. A perfect power spectrum would be completely flat at all frequencies. The power spectrum of white noise is flat, i.e. it contains an equal amount of all frequencies. The plots in this section have been created using the Matlab script residual_analysis.m. This can be viewed in App. B.5.

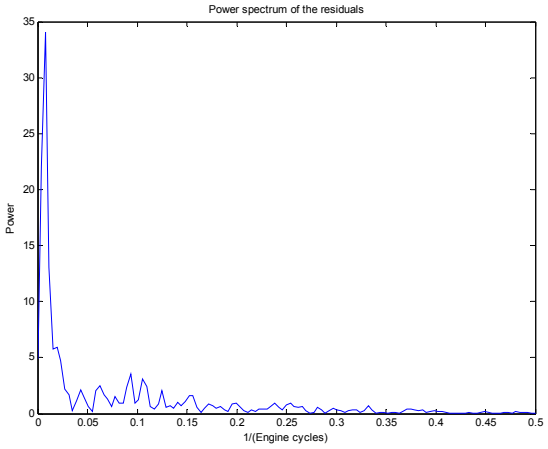

Figure 21 Power spectrum of the residuals when comparing measured and simulated output at operating point 17.

Obviously the model is not perfect at this operating point. Especially there is a spike at 0.01 cycles^{-1}. This might indicate that the system contains unmodelled dynamics. To check if this is the case, the same test is

[1] The residual is ideally white noise, however it can be contaminated by interference from other sources; e.g. power supply.

performed on the data in the file indexed 24. This is the same operating point, but the experiment is performed three days later. The result of this can be seen in Figure 22. The spike at 0.01 cycles^{-1} is present in this plot as well. However the plots are not identical, this indicates that the plots are partly the result of noise.

The unmodelled dynamics are at a very low frequency; this indicates that these dynamics can be effectively counteracted by an integrator.

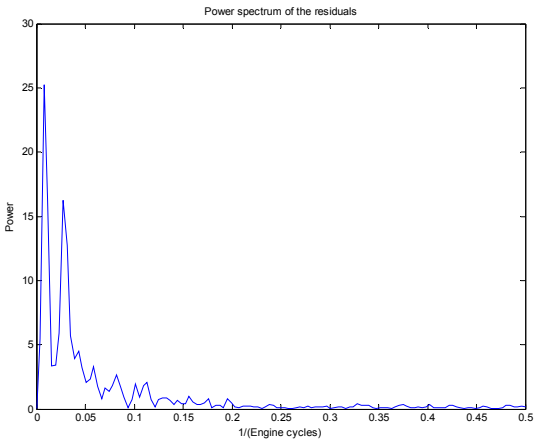

Figure 22 Power spectrum of the residuals when comparing measured and simulated output at operating point 24.

The reason for the non-perfect shape of the residual power spectra is likely to largely depend on random disturbances of the real process or from added effect of the PRBS-disturbances used to excite the process.

5.9 Model and Simulink

As stated in the introduction one of the objectives in this work is to create a Simulink block that can be used for simulations. This is useful when constructing controllers, since it is possible to implement controllers in Matlab and then test them against the simulated engine. This is of course preferable when a new controller is developed. There are several reasons why this is so.

It is easy to implement a new controller in Matlab since it contains functions for calculation of controller structures. The structures obtained can then be used directly to control the process.

A poor controller design might potentially harm the real engine if it steers the engine into regions where it is not designed to be. This will of course not harm the model. This means that the engineer constructing a new controller has greater freedom when tuning a new controller.

5.9.1 Constructing an engine block in Simulink

The term engine block in this case denotes a Matlab-Simulink block simulating the behaviour of the real engine, or rather one cylinder. To make an engine block in Simulink it is necessary to create what is called an S-function. An S-function can be written either as a Matlab function or as a function in some programming language such as C. For a description on how to write S-functions; see the Mathworks web site (www.mathworks.com).

For use in the continued work in this master thesis an m-file S-function has been created. The file is called HCCI.m and can be viewed in section B.7. This file describes how one cylinder of the engine responds to different input signals. In Simulink the file is masked with an interface where it is possible to enter the matrices describing the model. The interface can be viewed in Figure 23. To make the model mimic the behaviour of the real engine, noise has been added to the process states as well as the process output. It is possible to change the noise variance of the process and measurement noise individually via the interface.

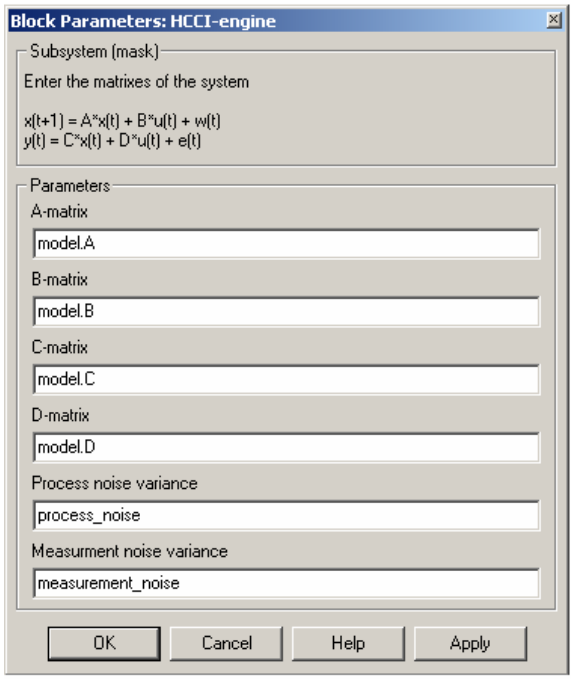

Figure 23 Interface of the engine Simulink block.

6 Constructing a controller

Now that a model of the real process is present it is possible to make an attempt at designing an effective controller of the process. For the task an LQG controller structure is chosen. The LQG structure is chosen because it handles MIMO systems, and because it is good at handling process and measurement noise. For more information on LQG control see (Åström and Wittenmark, 1997).

In this section the controller will work with a model that does not contain any delays. Delays will however be added to the control loop. The reason for this is to verify that the controller manages to perform well even though delays are not modelled perfectly.

6.1 Method

The task of constructing the controller is broken down into small steps. The first step is to construct an LQG controller based on the model obtained in the system identification part of this thesis. At this point the work is focused on making the output signal of the simulated engine follow a reference signal. In other words this is considered to be a servo problem (as opposed to a controller problem where the goal is to drive the output to zero). In this step there is no process or measurement noise added in the engine block.

In the second step both process and measurement noise is added to the process. The controller is tuned to deal with the now noisy process. An attempt is made to make the noise in the simulated process similar to the noise in the real process. This is done with regard to frequency content as well as for amplitude.

The third step is to make the process even more like the real process. This is done by adding delays to the control signals. Even though the model (as well as the real process) reacts immediately to changes in CR and IAT, it takes a few engine cycles before the ordered CR or IAT is achieved.

The fourth step is to verify that the controller manages to control the engine even though the process deviates from the model used to create the controller. If the controller is to be capable of controlling the real engine it has to perform well even though the model does not look exactly like the one the controller was designed for.

So far all work has been aimed at one cylinder and not the entire engine. Since both CR and IAT affect all cylinders in parallel the controller will have to work with mean values from the five cylinders. However it is interesting to find out if the controller developed here can be converted to control cylinder individual IAT. This will be done at the end of this section.

The controller is created using the Matlab commands *kalman*, *dlqr* and *lqgReg*. An example on how to use them can be found in section B.10.

The controller is not allowed to control the engine speed or fuel amount injected. These inputs will be a result of the conditions under which the engine is running. Thus the model of the engine is altered before feeding it to the Matlab functions used to create the

controller. The model fed to the *dlqr* function has had the B matrix altered to (see Ap. B.10):

$$B = \begin{pmatrix} 6.69\cdot10^{-3} & 2.38\cdot10^{-3} & 0 & 0 \\ 2.61\cdot10^{-3} & 1.26\cdot10^{-3} & 0 & 0 \\ -2.75\cdot10^{-3} & -1.88\cdot10^{-3} & 0 & 0 \end{pmatrix}$$

This change hides the control signals engine speed and load as far as the controller is concerned.

6.2 A comment on the Figures

Many of the Figures in this section will contain plots of the control signals (CR and IAT). To save space sometimes they are plotted in the same plot. In these plots the IAT signal is always the one with the largest fluctuations. One unit on the y-axis in these plots represent either 1° C or 1 CR unit.

The control signals are plotted to illustrate that there are no oscillations or large spikes. These combined plots are sufficient in this regard.

6.3 Step one, no noise

The thing to do is to create a Simulink model where it is possible to test the developed controller. The model can be viewed in Figure 24.

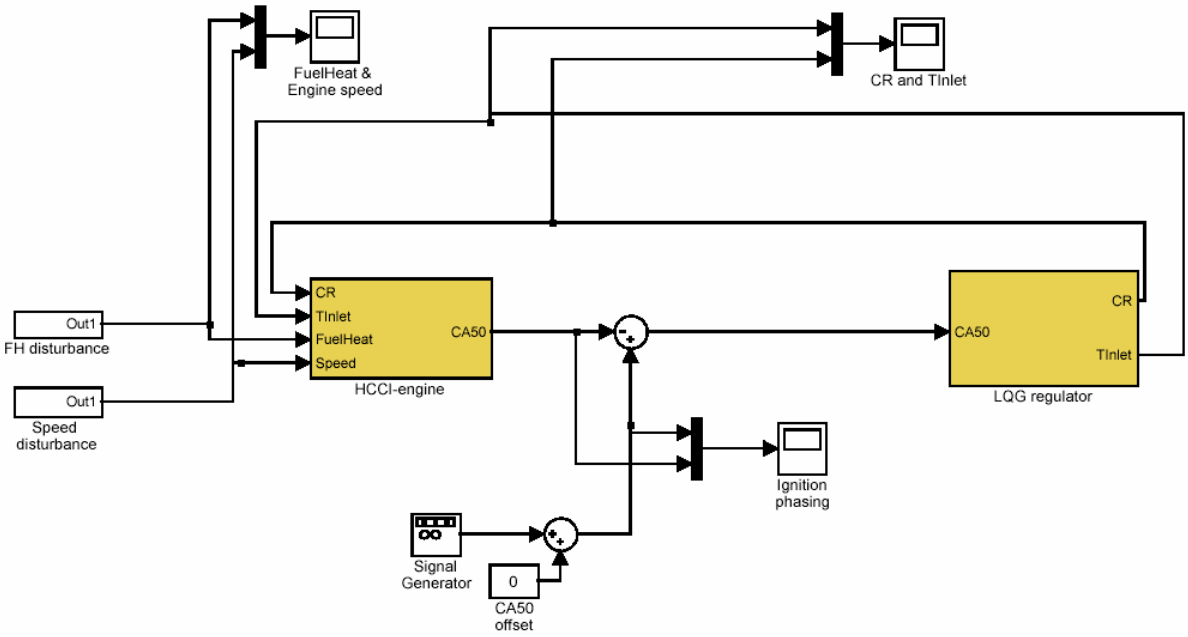

Figure 24 The Simulink model where the controller can be tested.

The two leftmost blocks in the Figure are input signals for fuel amount and engine speed. These input signals can be chosen to be constants with or without sinusoidal disturbances added. In steps one to three, CR has an offset of 270 J/cycle and speed 3000 rpm. The CA50 reference signal is always centred on 0.

When performing the disturbance rejection tests a sinusoidal signal with period of 200 cycles and amplitude of ±30 J/cycle is added to the fuel heat input. For the Engine speed the added disturbance has period 500 cycles and amplitude ±300 rpm.

When performing the reference following tests, no disturbances are added to CR and engine speed. Instead a square wave with a period of 1000/3 cycles and amplitude ±1° is added to the CA50 reference signal.

The block named HCCI-engine is a mask for the S-function HCCI.m that can be studied

in App. B.7. The mask is a user interface where it is possible to enter the values of the model as state space matrices. The interface can be studied in Figure 23.

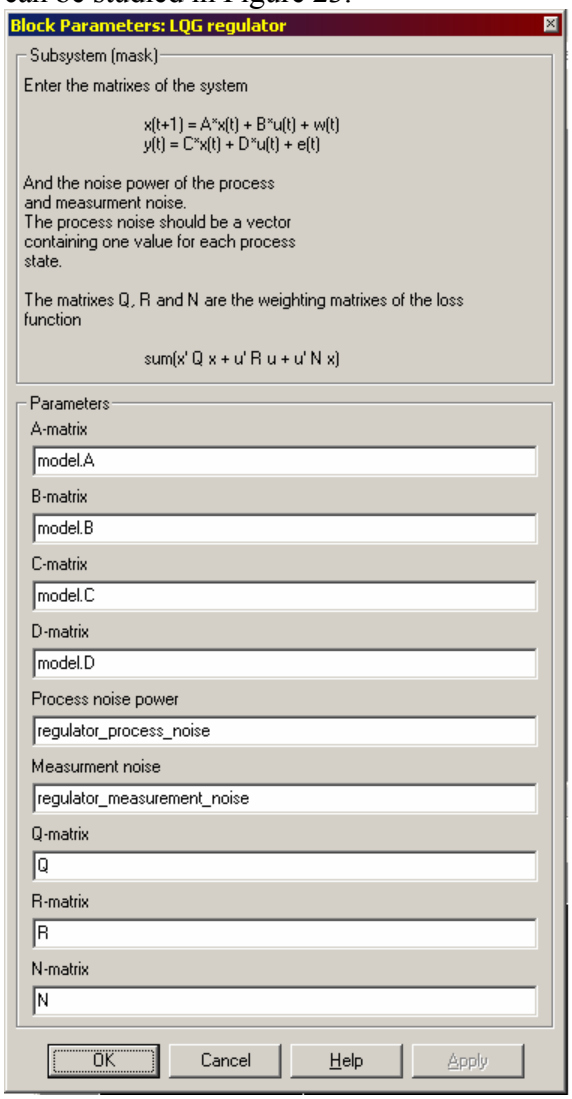

Figure 25 User interface for the LQG-controller mask. This interface is used to enter data used to create an LQG-controller.

The output of the engine block is the ignition phasing. Since all offsets are removed from the experiment data before system identification, the value 0° ATDC in the model actually corresponds to 4° ATDC on the real engine. The reason is that this is the mean offset of the experiment data. Since this is considered to be a servo problem the output is then subtracted from the reference value before the value is fed to the LQG-controller.

Just as with the engine block the LQG-controller block is a mask making it possible for the user to enter input data for the controller. The interface can be studied in Figure 25. As is the case with the engine block, the controller is an m-file S-function. This file can be viewed in App. B.11.

6.3.1 Result

The first attempt at making a controller proves to be a great disappointment. The output from the engine model when controlled by the LQG-controller in the set up described in Figure 24 does not perform well at all. The output of the first test can be viewed in Figure 26 and Figure 27. As can be seen in the plot, the controller is not capable of suppressing the disturbances. It is not even capable of following the reference signal when there are no disturbances.

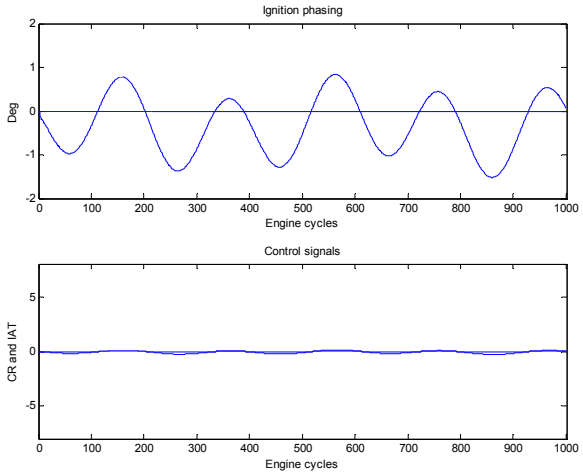

Figure 26 Ignition phasing (Upper plot) and control signals (lower plot) when performing a disturbance rejection test.

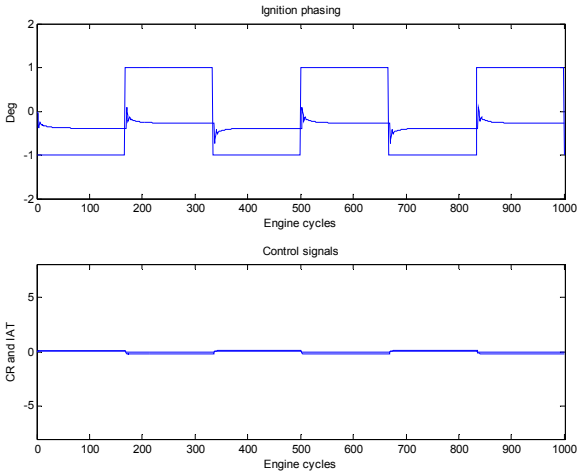

Figure 27 Ignition phasing (upper plot) and control signals (lower plot) when performing a reference following test.

The process has zeros close to 1 on the real axis. This means that the process is derivative in its behaviour. Adding an integrator to the controller can counteract this behaviour.

To solve the problem of the poor reference signal following it is decided to add an integrator acting on the error, i.e. on the difference between the reference signal and the process output. The Simulink set up is now as in Figure 28.

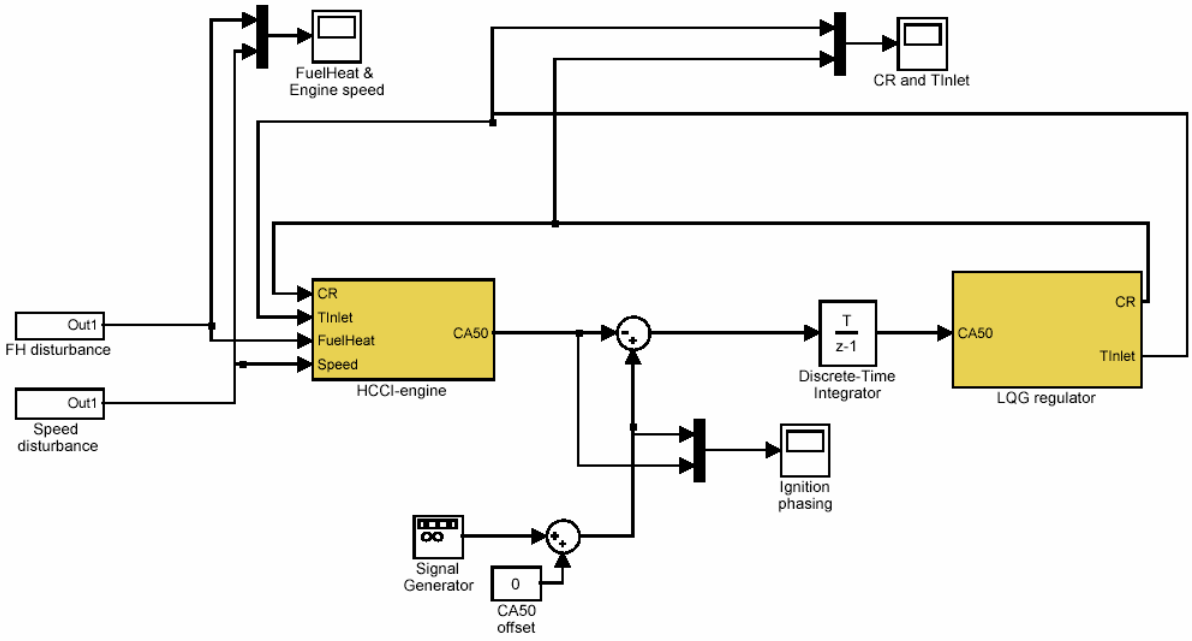

Figure 28 Simulink model after adding the integrator.

This set up proves to be superior to the first attempt. This controller manages to suppress the disturbances to the input signals better than the first attempt. The output is now within one deg. of zero at all times (see Figure 29 and Figure 30). It also manages to follow the reference signal very well when there are no disturbances. There is still room for improvements, but it is already performing fairly well.

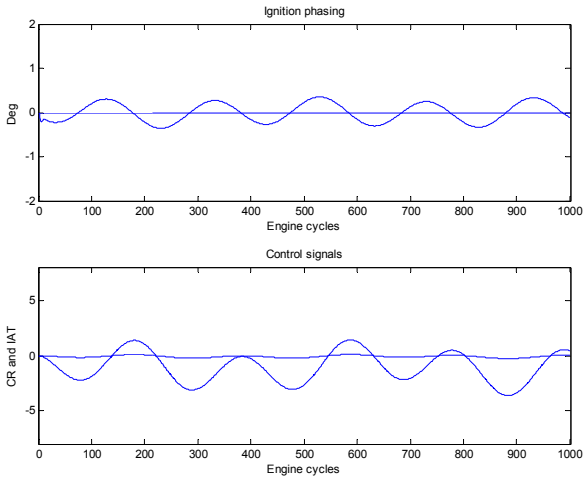

Figure 29 Ignition phasing (upper plot) and control signals (lower plots) when performing a disturbance rejection test.

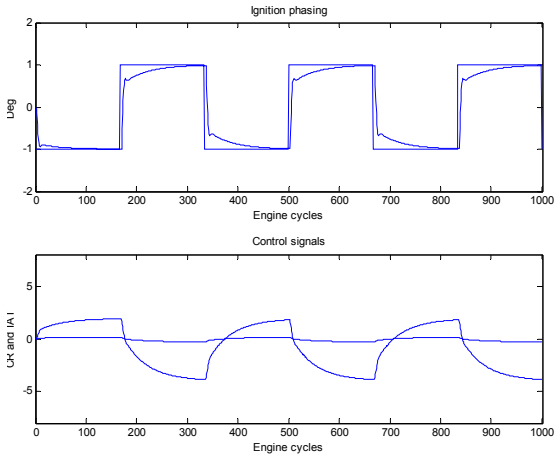

Figure 30 Ignition phasing (upper plot) and control signals when performing a reference following test

6.4 Step two, adding noise

Before adding noise to the engine model it is necessary to find out how much noise to add. For this purpose the output from the experiment indexed 3 is examined. This is a step disturbance experiment, but the step disturbances end after 400 engine cycles, and the output after this is pure noise. The reason for choosing this experiment is that the variable compression is positioned at one extreme; this minimizes the influence of limit cycles (Slotine and Li 1991) due to friction in the variable compression. The output signal used for this analysis can be viewed in Figure 31.

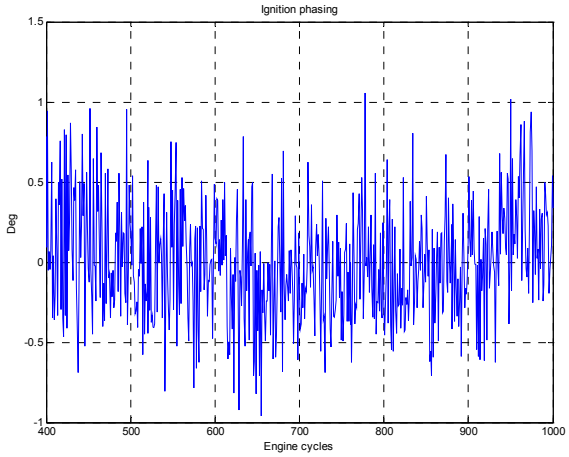

Figure 31 Measured output after cycle 400 of experiment indexed 3

There is a small drift in the measured output. This however should not cause any problems when evaluating the noise properties. The variance of the noise is calculated using Matlab command *var*, and the result is a noise variance of 0.128.

To see if the noise is white or coloured, the power spectrum of the noise is plotted. The power spectrum (Figure 32) is basically flat and has no dominant peaks. This leads to the conclusion that the noise is white.

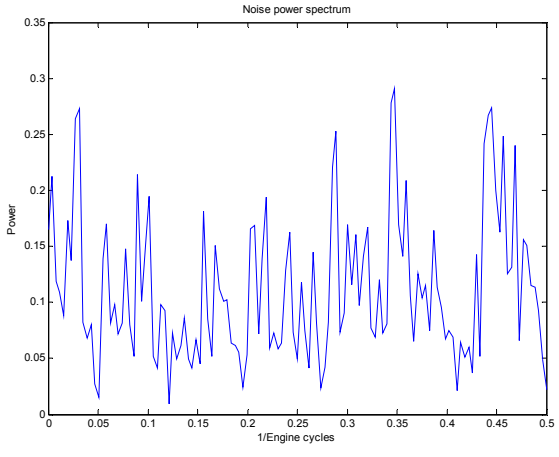

Figure 32 Power spectrum of the measured noise.

The noise plot, as well as the power spectrum and calculation of the noise variation is produced using Matlab script

test_noise.m, which can be viewed in App. B.12.

Trial and error suggests a process noise variance of $2*10^{-8}$ and a measurement noise variance of 0.125. This amounts to a variance of 0.135 for the simulated engine, and a noise spectrum (see Figure 33 and Figure 35) that is comparable to that obtained from the measured noise.

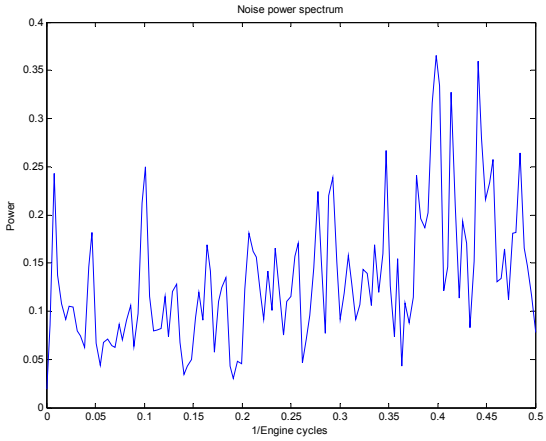

Figure 33 Power spectrum of the noise from the Simulink engine block.

6.4.1 Result

The controller is now working against a modelled engine containing both process and measurement noise. The output (Figure 34) is still very good. The controller is still working with a perfect model of the engine block including its noise properties. In other words, the controller should do well under circumstances like these.

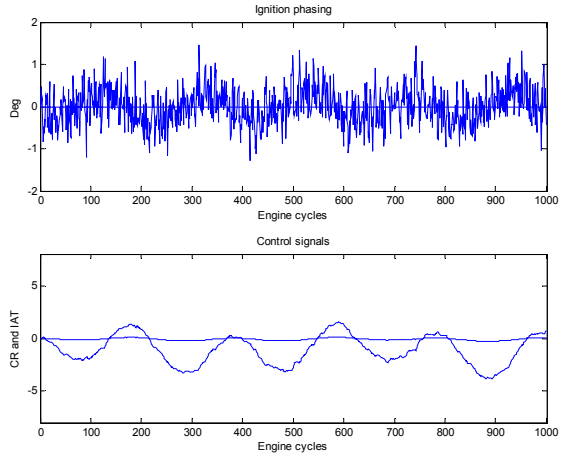

Figure 34 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment.

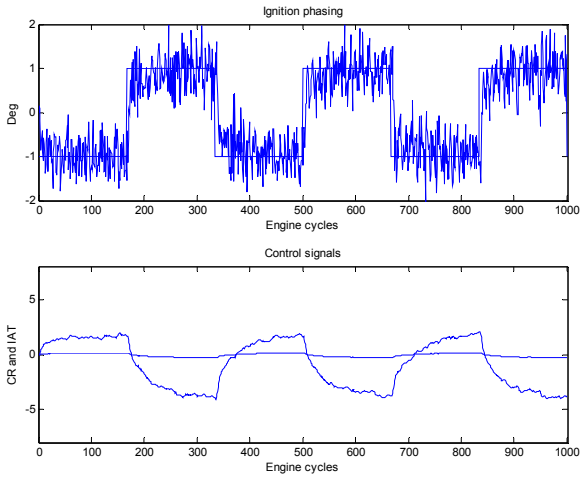

Figure 35 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment.

6.5 Step three, adding delays

Now that the controller has proved to be working under nearly optimal operating conditions with a perfect model and no delays, it is time to add delays to the control signals from the controller to the engine. This aims at making the simulated controller task resemble the task of controlling the real engine.

A study of the delay from a reference change to IAT to a noticeable change in CA50 indicates a delay of three engine cycles. Since the engine in itself contains a delay of one engine cycle, a delay of two cycles is

added to the CR and IAT signals going from the controller to the engine block.

6.5.1 Result

With the same controller settings as used to create Figure 34 and Figure 35 the output looks the same (Figure 36 and Figure 37). This can probably be explained by the controller being quite conservatively tuned in addition to using a perfect model.

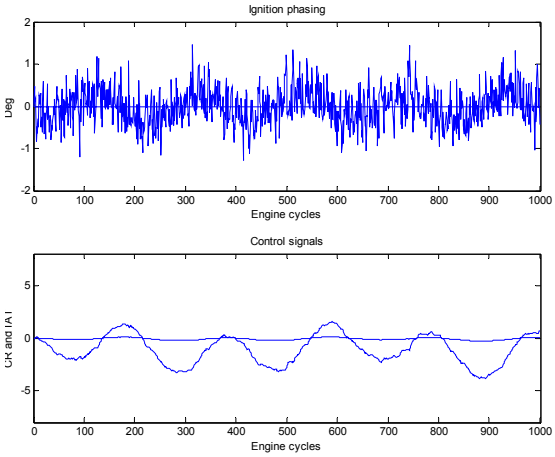

Figure 36 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment.

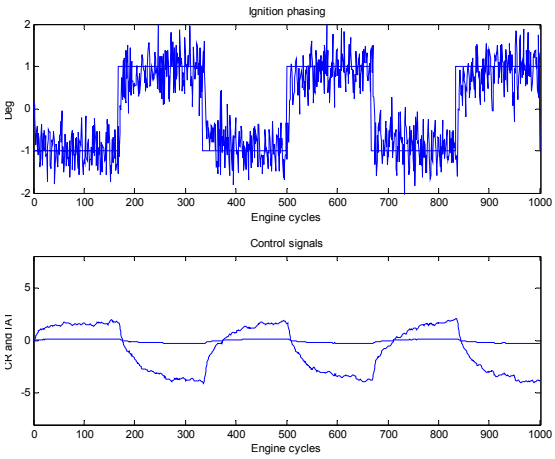

Figure 37 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment.

6.6 Step four, non perfect model

In the work so far the controller has been constructed using a perfect process model. This is of course highly unrealistic. The main purpose so far has been to verify that the chosen controller structure has potential to control the engine. And so far it has performed reasonably well.

In this step the controller will be tested against engine models optimised for one specific operating point. This means that the model of the engine now performs very much like the real engine would at a specific operating point. To keep the engine speed and load in the desired region, these signals are centred on the same operating points used to create the models. Added to them are sinusoid disturbances.

So far the behaviour of the controller has not been examined very closely. The tests so far have only aimed at verifying that the controller structure is capable of controlling the engine model. However it is equally important that the controller is not too aggressive. Aggressive in this context means that the controller tries too hard to make the output follow the reference signal. A too aggressive controller will cause excessive wear on actuator valves as well as cause the output to oscillate. The oscillations are partly caused by the delays between controller and engine. In this stage an attempt at optimising the controller for good performance without making it too aggressive is made.

6.6.1 Targeted operating points

The models used in this step are created using data from files indexed 23 to 29 (see A.4). These operating points cover a large part of the operating range of the engine. If a controller can be developed that works well on all of these operating points, that controller will be likely to perform well on the real process. The offsets and perturbation amplitudes used in the experiments are listed in Table 3. The sinusoidal perturbations have a period of 200 engine cycles for the load and 500 engine cycles for the speed.

Table 3 Engine speeds and loads used in the tests.

Operating point id.	Engine speed (rpm)	Engine load (J/cycle)
23	2000 ±300	240 ±30
24	2000 ±300	310 ±30
25	3000 ±300	250 ±30
26	3000 ±300	300 ±30
27	4000 ±300	240 ±30
28	4000 ±300	300 ±30
29	5000 ±300	240 ±30

6.6.2 Tuning the controller

LQG controllers aim at minimizing the loss function

$$J = \sum (x^T Q x + u^T R u + 2 x^T N u) \qquad (5)$$

where the weighting matrices Q, R and N are at least positive semi definite.

The Q matrix should be chosen as C^TC where C is the C matrix of the model to be controlled. N is usually chosen to be zero. The R-matrix is chosen relative in size to the Q-matrix. A large value on the Q-matrix compared to the R-matrix means that the controller mainly aims at minimizing the variance of the process output at the expense of the control signals. The opposite is true if the R-matrix is large compared to the Q-matrix. The N-matrix indicates that the controller should strive to minimize the covariance of the control signal and process output.

In addition to the Q, R and N matrices, the controller design procedure needs the process and measurement noise of the process to control. These do not necessarily have to be the correct values. A large value on the measurement noise indicates that the controller should be conservative in its response to changes in the process output. A large value on the process noise indicates that the controller should be conservative in its response to changes in the process states.

6.6.3 Result

Optimisation by trial and error indicates that the controller performs well on all operating points when it is designed using the following parameters:

$$w = \begin{pmatrix} 10^{-4} \\ 10^{-4} \\ 10^{-4} \end{pmatrix}$$

$$e = 5$$

$$Q = C^T C$$

$$R = \begin{pmatrix} 150 & 0 \\ 0 & 3 \end{pmatrix}$$

$$N = \mathbf{0}$$

where w denotes process noise and e measurement noise. Using the mentioned parameters generates a controller

$$x(t+1) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

where u(t) is the integrated difference between $CA50_{ref}$ and CA50 at time t and y(t) is a vector containing CR and IAT settings to be used as inputs to the engine. The x-vector contains the controller states. The matrices have the following values:

$$A = \begin{pmatrix} 0.3753 & 0.4094 & -0.4646 \\ -0.08874 & 0.5291 & 0.6581 \\ 0.2114 & 0.1456 & 0.5244 \end{pmatrix}$$

$$B = \begin{pmatrix} 9.413 \cdot 10^{-5} \\ 1.095 \cdot 10^{-4} \\ 9.134 \cdot 10^{-5} \end{pmatrix}$$

$$C = \begin{pmatrix} -1.281 & -3.316 & -0.9619 \\ -18.7 & -50.09 & -2.442 \end{pmatrix}$$

$$D = \begin{pmatrix} 7.817 \cdot 10^{-4} \\ -0.01091 \end{pmatrix}$$

With the settings mentioned above the simulation results are as indicated in Figure 38 to Figure 51. In the mentioned Figures the simulated output as well as the control signals used to obtain them can be viewed.

The results for all tested operating points are acceptable. The process is noisy but the control signals are still relatively smooth. This is important since it means that the

actuators are not excessively worn. Further optimisation work might result in better performance. However this will not be done at this point.

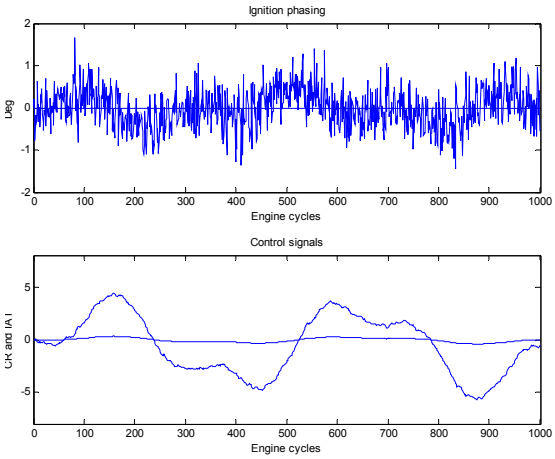

Figure 38 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment. Engine model used is optimised for op. 23.

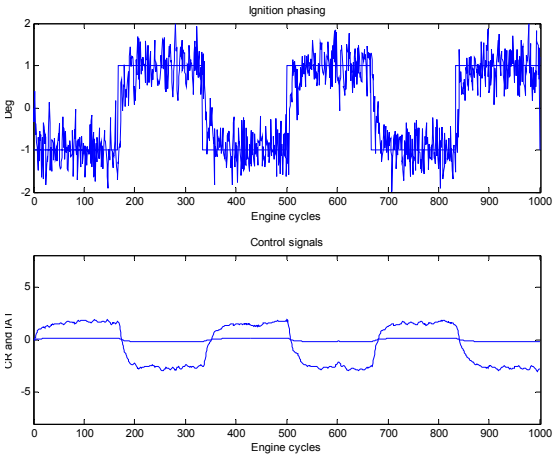

Figure 39 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment. Engine model used is optimised for op. 23.

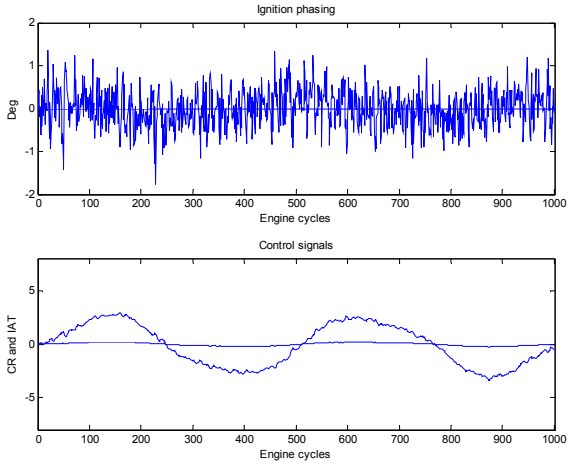

Figure 40 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment. Engine model used is optimised for op. 24.

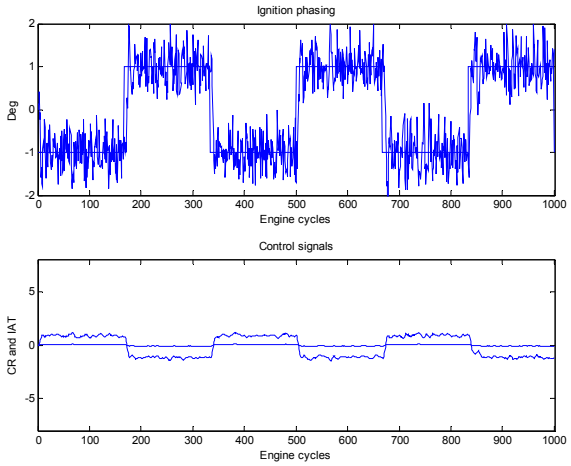

Figure 41 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment. Engine model used is optimised for op. 24.

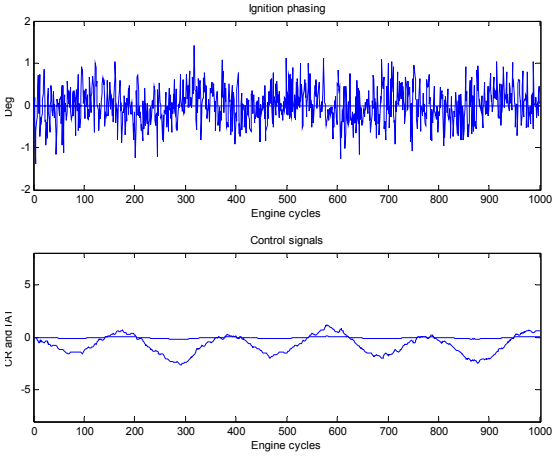

Figure 42 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment. Engine model used is optimised for op. 25.

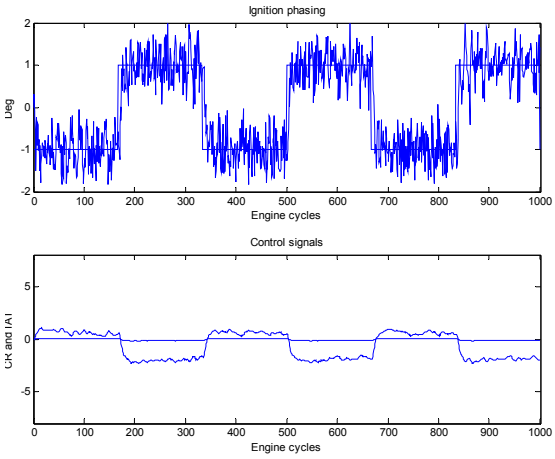

Figure 43 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment. Engine model used is optimised for op. 25.

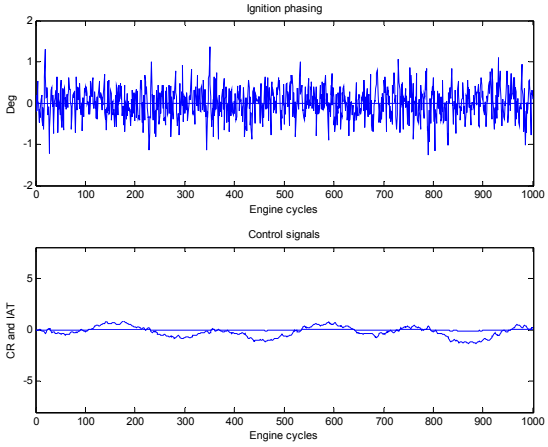

Figure 44 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment. Engine model used is optimised for op. 26.

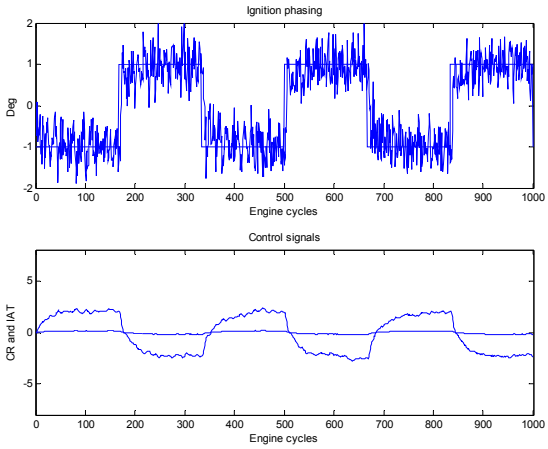

Figure 45 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment. Engine model used is optimised for op. 26.

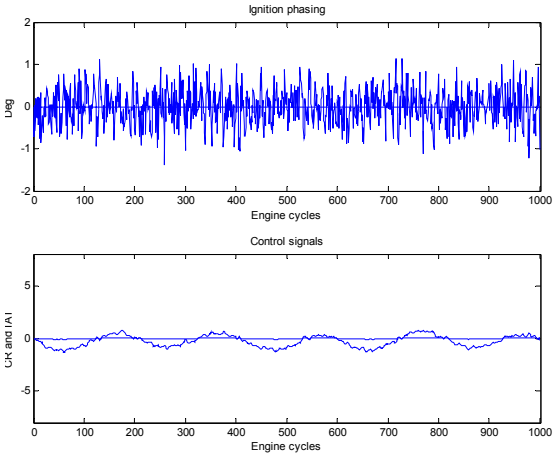

Figure 46 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment. Engine model used is optimised for op. 27.

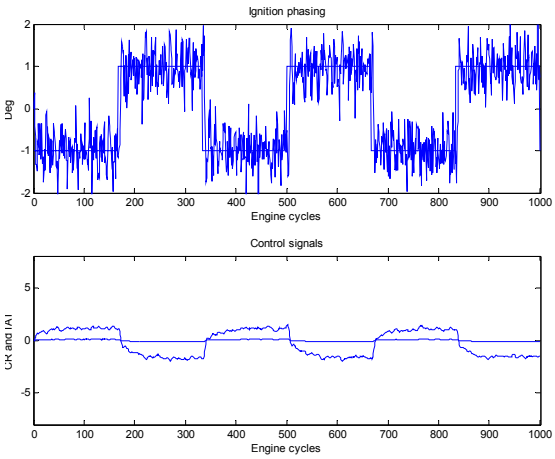

Figure 47 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment. Engine model used is optimised for op. 27.

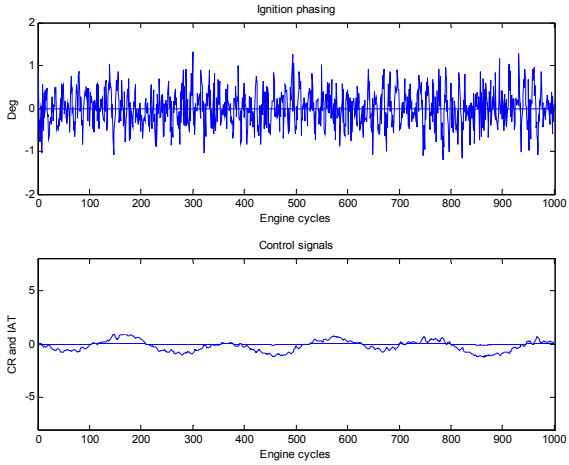

Figure 48 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment. Engine model used is optimised for op. 28.

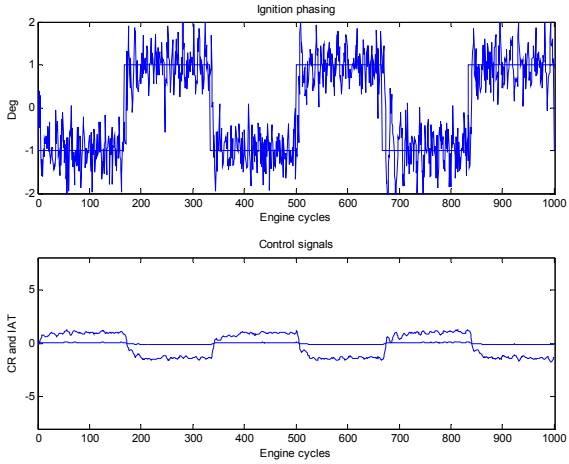

Figure 49 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment. Engine model used is optimised for op. 28.

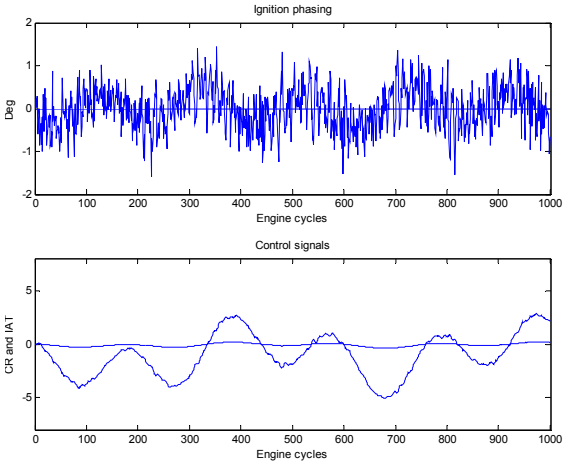

Figure 50 Ignition phasing (upper plot) and control signals (lower plot) for disturbance rejection experiment. Engine model used is optimised for op. 29.

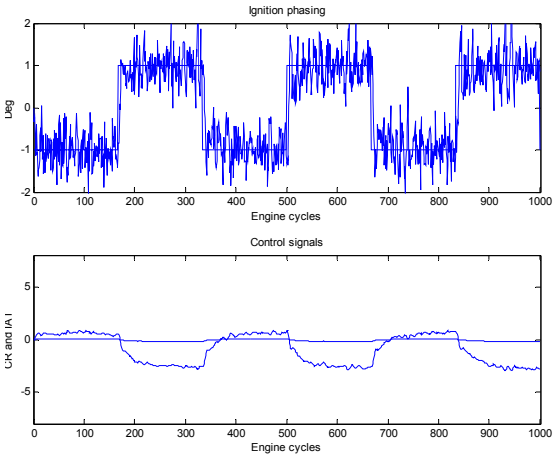

Figure 51 Ignition phasing (upper plot) and control signals (lower plot) for reference following experiment. Engine model used is optimised for op. 29.

The simulations so far have been on one cylinder only. To make the controller valid for the complete five-cylinder engine is an easy task however. IAT and CR will then be based on a mean value of CA50 from each of the individual cylinders. This method is used when performing the tests in section 6.8.

6.7 Sensitivity function

The sensitivity function is the transfer function from e to y in Figure 52. This is a measure of how well the controller manages to suppress load disturbances.

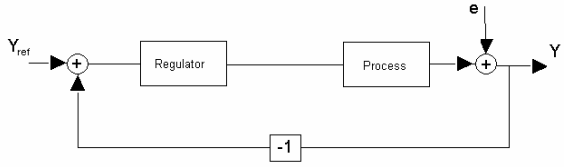

Figure 52 Set up of model and controller.

The Bode plot (see Figure 53) indicates that the controller manages to suppress disturbances well in the mid range. The controller does not counteract high frequency disturbances. This is actually the desired behaviour of the controller since this would lead to excessive wear on the actuators.

The controller does not manage to suppress really low frequency disturbances. It is likely that the reason for this is that the process contains zeros close to 1 on the real axis. These zeros cancel out the integrator in the controller.

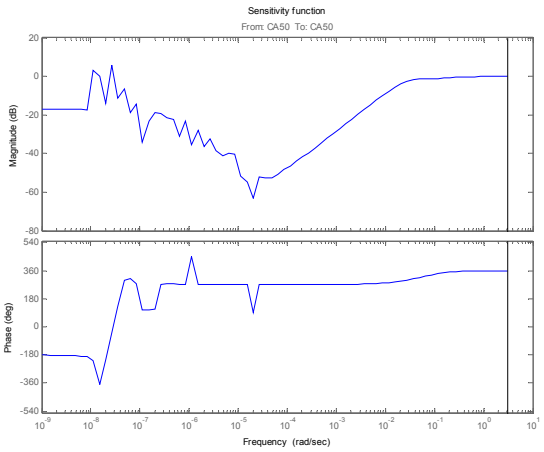

Figure 53 Bode plot of the sensitivity function.

6.8 Performance: LQG vs. PID

As mentioned previously the ignition phasing is presently controlled using PID controllers of the IAT and CR. The set up currently used only allows for one of the input signals at a time to be used to control the engine. If the developed controller cannot perform at least equally well as the

existing controller there will be no use for it. Thus their performances are compared here.

The ability of the controller to make the process follow a reference signal, as well as suppressing perturbations to engine speed and load is examined for a PID controller as well as for the developed LQG controller.

In section 6.6.3 it can be seen that the LQG controller performs worse at operating point indexed 23 than it does at any other tested operating point. Thus the performance of the LQG controller is compared against a PID controller optimised for this operating point. The LQG controller is not optimised for this operating point. This means that if the LQG controller manages to perform equally well as the PID at this point, it is likely to perform equally well or better at all other operating points. The noise model for the engine is the same as the one obtained in section 6.4.

6.8.1 Method

The first step is to compare the two controllers' ability to suppress disturbances to the engine speed and load. In this test the CA50 reference signal is constant zero. A sinusoidal disturbance with amplitude ±30 J/cycle and period 200 cycles is added to the offset value 240 J/cycle for the engine load. A sinusoidal disturbance with amplitude ±300 rpm and period 500 cycles is added to the offset value 2000 rpm for the engine speed.

The second step is to compare how well the controllers can make the engine output follow a reference signal. In this test no perturbation is added to the engine speed and load. The constant offsets are the same as for the disturbance rejection test. The ignition phasing reference signal is a square wave centred on 0 with amplitude ±1° and period 1000/3 cycles

In this test the PID controller is using only IAT to control the process since this is the way it is done on the real engine.

The engine model used in this experiment is a five-cylinder model. The output from the model is considered to be a mean of the output from each of the five cylinders. No cylinder balancing is used. The plots show the mean of the output for the five cylinders.

6.8.2 The LQG controller

The LQG controller used in this test is the same as the one obtained in section 6.6.3. The input to the controller will be a mean of the ignition phasing for the five cylinders. The same control signals will then be sent to each of the individual cylinders.

6.8.3 The PID controller

The PID controller used in these tests use Eq. (6), where K_P=0.2, K_I=0.35 and K_D=0.15. Optimisation of the PID parameters is performed using trial and error. The parameters used in the controller of the real engine can not be used here since they are designed to control the IAT actuator, and not the temperature. This is the controller structure used on the real engine. The input to the controller is a mean of the ignition phasing for the five cylinders.

$$\begin{aligned} u(t) &= P(t) + I(t) + D(t) \\ P(t) &= K_P(y_{ref}(t) - y(t)) \\ I(t) &= I(t-1) + K_I(y_{ref}(t) - y(t)) \\ D(t) &= K_D(y(t) - y(t-1)) \end{aligned} \tag{6}$$

6.8.4 Disturbance rejection, result

The results of the disturbance rejection tests can be viewed in Figure 54 and Figure 55. The PID controller manages to keep the ignition phasing within one degree of the reference value. The LQG controller manages to keep the ignition phasing within 0.8 degrees of the reference value. In other words the LQG controller is 20% better at rejecting disturbances.

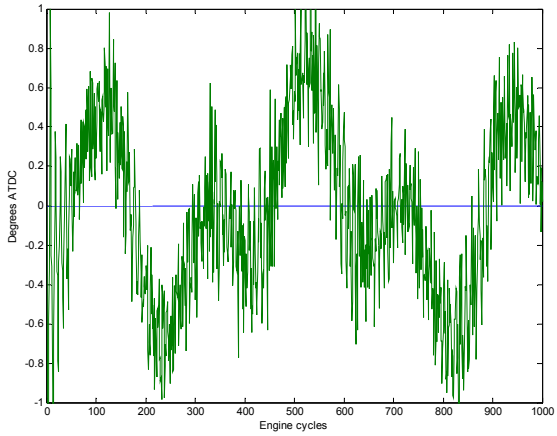

Figure 54 Disturbance rejection of the PID controller.

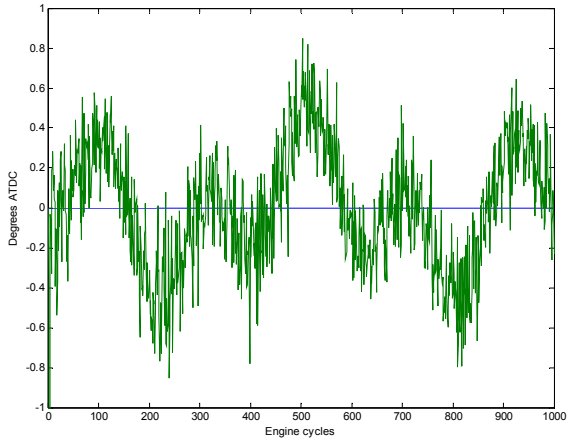

Figure 55 Disturbance rejection of the LQG controller.

6.8.5 Reference following, result

The results of the reference following tests can be viewed in Figure 56 and Figure 57. The PID controller manages to get the ignition phasing to a within 5% of the desired value within approximately 100 cycles after a step change. The LQG controller manages to get the ignition phasing within 5% of the desired value within approximately 80 cycles. Again the LQG controller outperforms the PID controller by 20%.

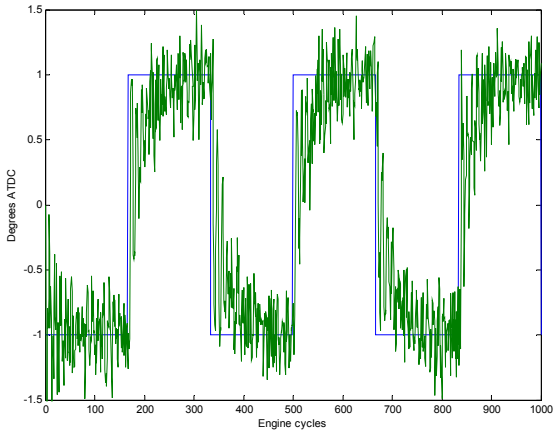

Figure 56 Reference following for the PID controller.

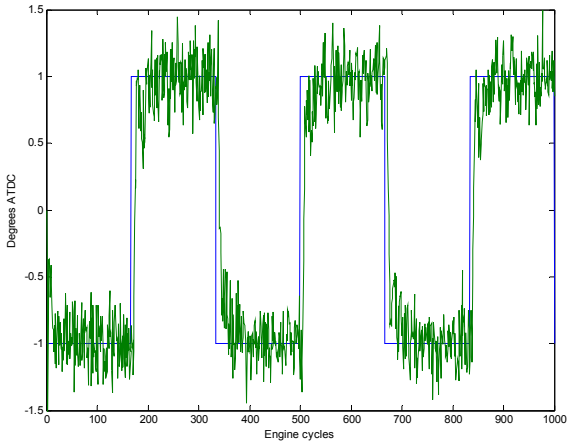

Figure 57 Reference following for the LQG controller.

6.9 Cylinder individual temperature control

In the work done so far it has been assumed that CR and IAT affect all cylinders in parallel. However it is planned that the engine examined in this thesis will have cylinder individual IAT control. An attempt is now made to convert the controller developed in the previous section to these circumstances.

The idea is to rewrite the controller to control IAT individually for each cylinder and CR for all cylinders in parallel. In the new controller the matrices will be called: $\tilde{A}, \tilde{B}, \tilde{C}, \tilde{D}, \tilde{x}$ *and* $\tilde{y}$. The new controller is built according to the following scheme:

$$\tilde{A} = \begin{pmatrix} A & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & A & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & A & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & A & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & A \end{pmatrix}$$

$$\tilde{B} = \begin{pmatrix} B & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & B & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & B & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & B & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & B \end{pmatrix}$$

$$\tilde{C} = \begin{pmatrix} C(1,:)/5 & C(1,:)/5 & C(1,:)/5 & C(1,:)/5 & C(1,:)/5 \\ C(2,:) & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & C(2,:) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & C(2,:) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & C(2,:) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & C(2,:) \end{pmatrix}$$

$$\tilde{D} = \begin{pmatrix} D(1)/5 & D(1)/5 & D(1)/5 & D(1)/5 & D(1)/5 \\ D(2) & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & D(2) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & D(2) & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & D(2) & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & D(2) \end{pmatrix}$$

$$\tilde{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$$

$$\tilde{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{pmatrix}$$

$$\tilde{y} = \begin{pmatrix} y^{cr} \\ y_1^{temp} \\ y_2^{temp} \\ y_3^{temp} \\ y_4^{temp} \\ y_5^{temp} \end{pmatrix}$$

using Matlab notation. A, B, C and D in the above matrices are the ones given in section 6.6.3. x_z and y_z^{temp} denotes controller states and temperature set point for cylinder z whereas y^{cr} denotes CR set point for the entire cylinder bank. u_z Denotes the integrated output error from the process for cylinder z. The extended controller structure now is

$$\begin{aligned} \tilde{x}(t+1) &= \tilde{A}\tilde{x}(t) + \tilde{B}\tilde{u}(t) \\ \tilde{y}(t) &= \tilde{C}\tilde{x}(t) + \tilde{D}\tilde{u}(t) \end{aligned} \qquad (7)$$

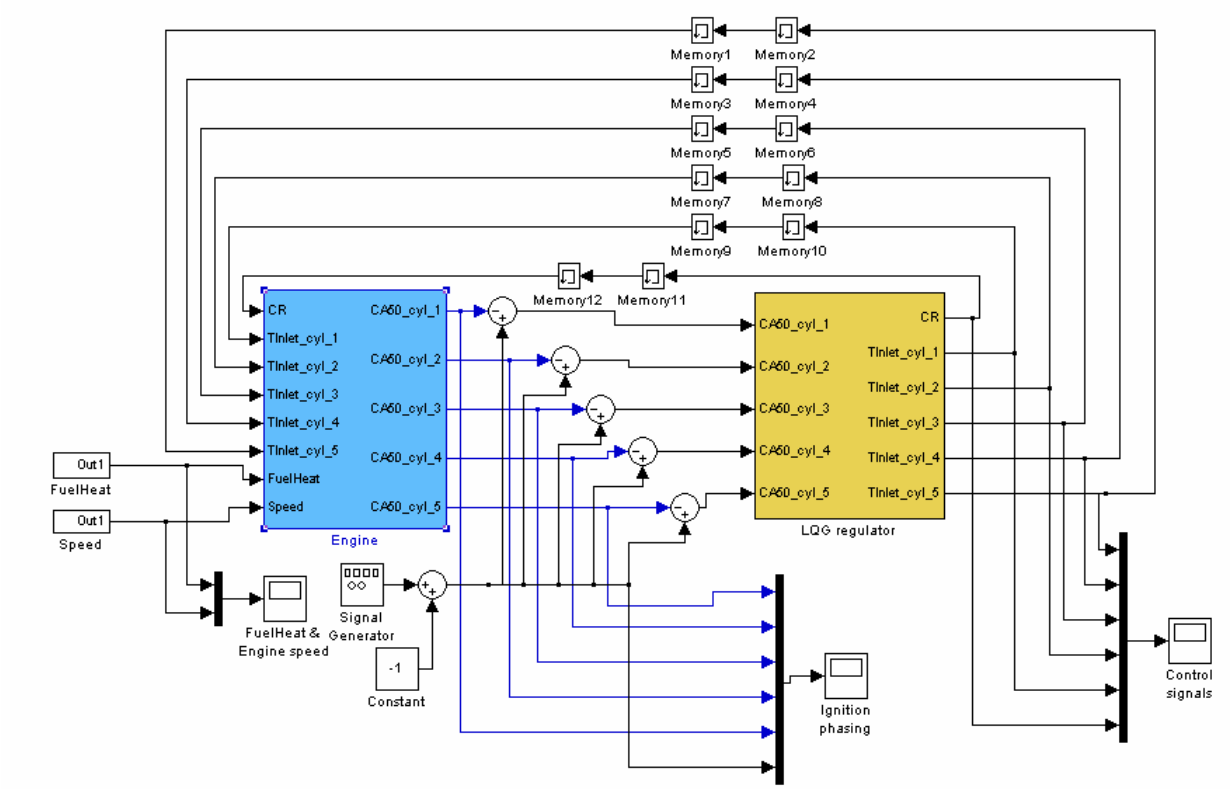

Figure 58 Picture of the Simulink model simulating the complete engine with cylinder individual IAT control. In this Simulink model the integrators has been included in the controller block.

The full model of the five-cylinder engine with the cylinder individual IAT control can be viewed in Figure 58.

Even though the cylinders on the real engine display little difference in their response to perturbations in the input signals there is still the matter of offsets in their outputs. These offsets are the result of different heat loss for the different cylinders. To verify if the suggested controller structure is capable of handling these differences a test is performed.

Each of the five cylinders in the engine model is supplied with a model optimised for different operating points; the operating points used are listed in Table 4.

Table 4 Models for the individual cylinders are optimised for the operating points listed here.

Cyl. nr.	Model optimised for op. index
1	29
2	28
3	27
4	26
5	25

6.9.1 Test set up

The test is performed using the set up and controller described in the previous section. The engine load used is 280 J/cycle and the engine speed is 3500 rpm. These values are chosen since they are in the centre of the examined load and speed ranges. In addition a sine perturbation with period 200 cycles and amplitude 30 J/cycle is added to the engine load. To the engine speed a sine perturbation with period 500 cycles and

amplitude 300 rpm is added. The ignition phasing reference signal is a square wave centred on zero with a period of 250 cycles and amplitude ±1.5°.

6.9.2 Result

The result of the test performed according to the previously described set up can be viewed in Figure 59 to Figure 64.

The results of the test are promising. The controller manages to keep the ignition phasing close to the reference signal for each of the cylinders even though it has to manage different models for each cylinder.

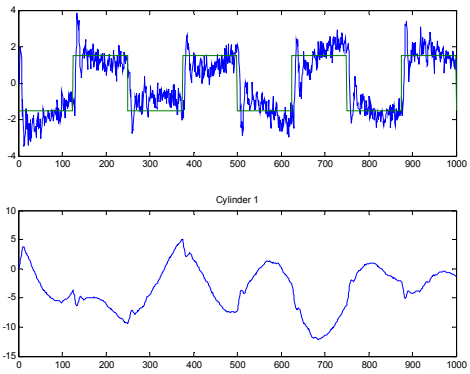

Figure 59 Ignition phasing and IAT for cylinder 1.

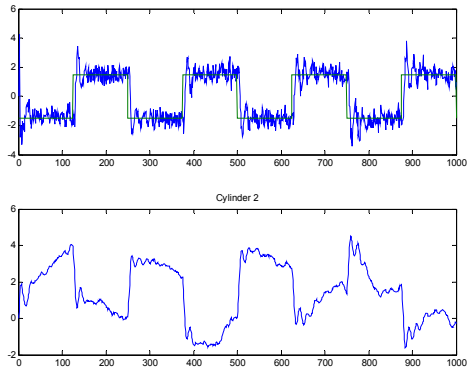

Figure 60 Ignition phasing and IAT for cylinder 2.

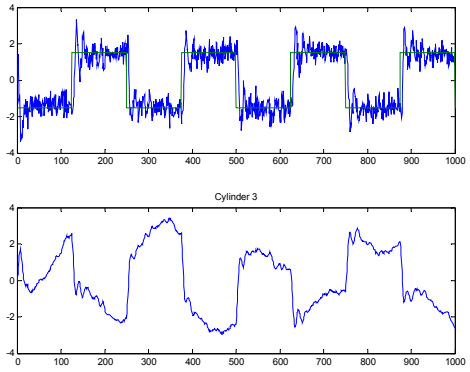

Figure 61 Ignition phasing and IAT for cylinder 3.

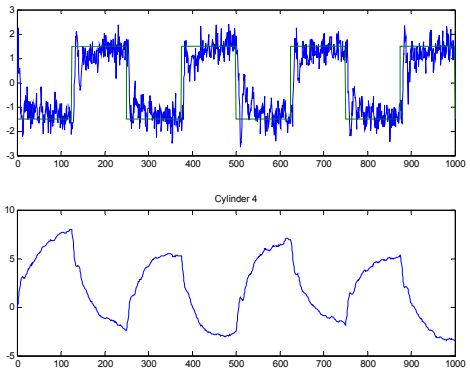

Figure 62 Ignition phasing and IAT for cylinder 4.

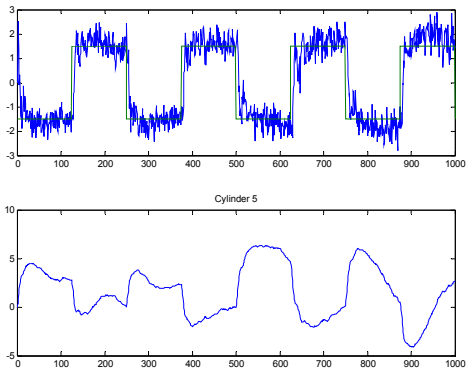

Figure 63 Ignition phasing and IAT for cylinder 5.

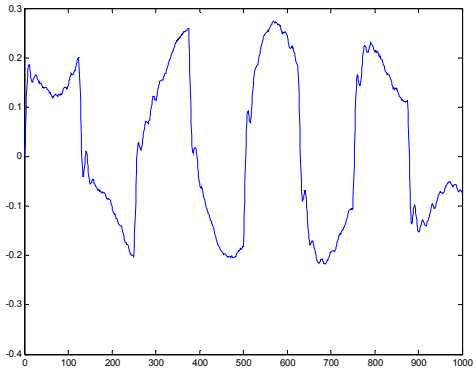

Figure 64 Compression ratio during the test using different models for each cylinder.

7 Implementation in Java

In this section the controller suggested in section 6.9 is implemented in Java. An attempt is made to make the suggested implementation independent of the size of the matrices of the controller.

7.1 Real time considerations

In section 6.9 the suggested controller calculates the control signals for all cylinders in parallel. This strategy works well in theory, but not so in practice.

If the engine is working at full speed (6000 rpm) the time for a complete engine cycle is only 2/6000*60=0.02 seconds. However the ignition of the five cylinders is evenly spread over the engine cycle, thus the time between the last ignition of an engine cycle to the first ignition of the next cycle is only 0.02/5=0.004 seconds. In this time measurement data has to be collected, computations made and signals sent to the actuators. It is obviously advantageous to have 0.02 second instead of 0.004 to perform these tasks. Thus the control problem is broken down so that it is possible to perform as much computations as possible as soon as the data is available.

7.2 Result

The suggested implementation can be viewed in App. B.14. This implementation demands that the controller has two outputs; namely CR and IAT, and that CR is controlled for all cylinders in parallel and that IAT is controlled individually for the different cylinders. The complexity of the controller and the number of inputs is arbitrary.

For the controller suggested in this paper the input to the controller should be the integrated difference between CA50 and $CA50_{ref}$. This paper gives no suggestion on how to write an integrator since this is very simple.

The Java file represents only the controller and not the interface between hardware and software. In other words; this file is intended to be used by some kind of intermediate interface.

8 Conclusions

The model obtained when performing system identification shows high correlation between simulated and measured output. This indicates that using several input signals to the engine when performing system identification is a good strategy.

The controller developed in this paper is very simple indeed. It does not model any delays. It does not make use of any form of gain scheduling. And it does not use feed forward. Even though it is so simple it performs well in the simulations. This indicates that this controller structure might be effective also on the real engine. It also indicates the LQG controller is the strategy to prefer when controlling the ignition phasing on the real engine. However, it is important to realise that the controller only has been tested against linearised models of the real process. There is no guarantee that it will work equally well on the real process.

9 Discussion

This section addresses some points that might benefit from a further investigation.

9.1 Increasing performance

It is quite possible that increased performance can be obtained by modelling the

delays. Modelling the delays is not hard; all that needs to be done is to add a few new states to the model, representing the delays.

Developing some kind of feed forward compensation for disturbances caused by changes in injected fuel amount and engine speed might increase the controller's ability to suppress disturbances.

The suggested methods to increase the performance of the controller lead to a more complex structure. This also means that there will be more parameters to tune; this might lead to a controller that is very hard to tune.

An advanced approach to reach high performance is to use adaptive control. This approach can be combined with the expanded models mentioned above.

Adaptive control has the potential to produce excellent results. However it does have it's drawbacks as well. One is that it is computationally heavy. Another is that it can be difficult to tune. Other problems might occur as well but are not listed here.

The PID controllers currently used on the engine use gain scheduling, this strategy is applicable to the LQG controller as well. This can be combined with all the performance increasing strategies mentioned above.

9.2 Variable compression ratio and friction

The variable CR is obtained by means of a system that allows the cylinder bank to lean more or less. This system works very well, however it is subject to friction. Limit cycles in the variable compression has been observed in initial experiments. These limit cycles are no major problem as far as the ignition phasing is concerned, but it is likely that this phenomenon will cause unnecessary wear on the mechanics involved.

9.3 Speed of the thermal element

Even though the thermal element has been changed to a faster one during the work on this paper, it is still quite slow. The IAT step response test (section 3.2 Figure 4) shows that the engine response does not match the temperature very well. If we instead look at the PWM signal to the valve controlling the inlet air temperature (Figure 65), and CA50 there is a very strong correlation. The reason is that the thermal element is too slow. Measuring air temperature fast is quite hard.

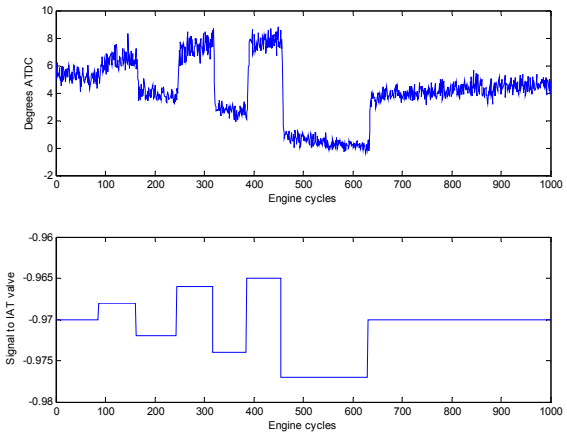

Figure 65 Ignition phasing (upper plot) and PWM signal (lower plot) for the temperature step response experiment. In this plot the PWM signal has been inverted to illustrate the close correlation between the signals.

In simulations temperature can be changed immediately thus the slow thermal element is not a problem. When it comes to controlling the real engine however it is likely that looking at the PWM instead will produce better results.

One difference now is that there will be a noticeable delay from a change in the PWM signal to a change in ignition phasing. When looking at the temperature there is no delay since the thermal element is slow.

9.4 Operating range

This paper addresses combustion control at engine speeds of at least 2000 rpm. A few experiments at idle speed have been performed. The results however have been very poor.

The work done so far indicates that the process is a lot harder to identify at very low engine speed than it is at speeds above 2000

rpm. The conclusion is that more work needs to be done to create a model for the engine at low engine speed.

In the experiments performed as part of this work the engine has been naturally aspirated. A comparison of the model obtained under these circumstances to a model or models valid for the super charged case would be interesting.

10 References

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

Ljung, L, (1999) *System Identification, Theory for the user,* Prentice Hall

Haraldsson, Göran, (2003) *Combustion Control of the Homogeneous Charge Compression Ignition engine* Thesis for the degree of Licentiate, LTH, ISRN: LUTMDN/TMHP--03/7010--SE

Olsson, Jan-Ola, (2002) *Performance and Control of the Homogeneous Charge Compression Ignition (HCCI) Engine* Thesis for the degree of Licentiate, LTH, ISRN: LUTMDN/TMHP--02/7002--SE

Olsson, Jan-Ola, et. al. (To be published at IFAC 2004) *Closed Loop Identification of the HCCI Process*

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

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

Bowns, D., (1971) *The Dynamic Transfer Characteristics of Reciprocating Engines*, Proc. Inst. Mech. Engrs., Vol. 185, pp.185-201.

Welbourn, D.B., Roberts, D.K., Fuller, R.A., (1959) *Governing of Compression-Ignition Oil Engines*, Proc. Inst. mech. Engrs., Vol. 173, pp. 575-604.

Åström, J. Karl and Wittenmark, Björn (1995) *Adaptive control,* Addison-Wessley Publishing Company

11 Acknowledgements

I would like to thank the people at the Division of Combustion Engines for their help. Göran Haraldsson helped me perform the practical experiments. Göran, Jari Hyvönen and Jan-Ola Olsson were valuable sources of ideas and support throughout my work.

From the department of Automatic Control I would like to thank Johan Bengtsson who helped me when I had control related problems.

I would also like to thank my supervisors Professor Rolf Johansson at the Department of Automatic Control and ass. Professor Per Tunestål at the department of Automatic Control for letting me do this work and for their inputs in the work of writing this paper.

12 Abbreviations

ATDC – After Top Dead Centre
TDC – Top Dead Centre
CR – Compression Ratio
IAT – Inlet Air Temperature
OP – Operating point
LQG – Linear Quadratic Gaussian
CA50 – Crank angle degree where 50% of the fuel has been consumed.
HCCI – Homogeneous Compression Charge Ignition
CI – Compression Ignition
SI – Spark ignition
LTH – Lunds Tekniska Högskola (Lund Institute of Technology)
PWM – Pulse Width Modulation
RPM – Revolutions Per Minute

A Operating points

In this Appendix the operating points of the test runs are listed.

A.1 Preliminary experiments

These are the preliminary experiments whose main objective is to find linearity regions. In the first tests the PWM signal was not saved since it was not known that the thermal element was to slow. After experiment indexed 6 this was corrected.

Id.	File name	Mean load (J/cycle)	CR	PWM	Mean temp.	Speed	CA50 mean	Comments
1	0306120936	220	17-19	?	224	2000	6.7	CR step changes
2	0306120954	220	21	?	140 - 175	2000	3.8	Temperature step changes
3	0306121008	180 - 280	21	?	159	2000	3.8	Load step changes.
4	0306121109	220	21	?	168	1500 - 4000	3.3	Speed step changes. Hot air valve 97% open
5	0306121121	220	21	?	243	1500 - 4000	3.7	Speed step changes. Hot air valve 100% open
6	0306121414	220	18.6 ±0.3	?	208	2027	3.9	CR excitation
7	0306121509	240	21	0.98 ±0.005	167	1999	4.1	PWM excitation

A.2 First set of main experiments

From now on the PWM signal is available in all files. The objective of these experiments is to find out what excitation levels can be used when excitation is applied to all input signals concurrently. Different PRBS frequencies are tested on the load excitation.

Id.	File name	Mean load (J/cycle)	CR	PWM (%)	Mean temp.	Speed	CA50 mean	Comments
8	0306171059	240	18 ±0.3	98	215	2001	4	CR excitation
9	0306171104	240	17.8 ±0.3	98	215	2002	4.7	CR excitation
10	0306171108	240	17.7	98 ±0.5	209	2003	6.2	PWM excitation
11	0306171113	240 ±40	17.6	98	219	2004	4.2	Load excitation, fast
12	0306171122	240 ±20	17.6 ±0.15	98 ±0.25	215	2003	5	50% excitation, Fast excitation of load
13	0306171128	240 ±60	17.6 ±0.45	98 ±0.75	217	2005	4.6	150% excitation, Fast excitation of load
14	0306171200	240 ±40	18.4	98	204	2005	4.8	Load excitation

A.3 Second set of main experiments

It is now decided to lower the excitation amplitude by 50% since the added effect of the PRBS signals to all the input signals causes the engine to misfire. From now on it is also possible to add PRBS disturbances to the engine speed. This set is used for system identification.

Id.	File name	Mean load (J/cycle)	CR	PWM (%)	Mean temp.	Speed	CA50 mean	Comments
15	0306240949	240 ±20	21+0 - 0.25	97.2 ±0.15	176	2000 ±200	3.9	Concurrent excitation
16	0306241002	240 ±20	20 ±0.15	97.45 ±0.25	192	2000 ±200	3	Concurrent excitation
17	0306241011	310 ±20	20 ±0.15	96.9 ±0.25	171	2000 ±200	3.7	Concurrent excitation
18	0306241145	250 ±20	20 ±0.15	96.9 ±0.25	189	3000 ±200	3.9	Concurrent excitation
19	0306241150	300 ±20	18.85 ±0.15	96.9 ±0.25	193	3000 ±200	4,7	Concurrent excitation
20	0306241154	240 ±20	18.5 ±0.15	96.9 ±0.25	203	4000 ±200	4.8	Concurrent excitation
21	0306241200	300 ±20	17.5 ±0.15	96.9 ±0.25	207	4000 ±200	5.7	Concurrent excitation

A.4 Third set of main experiments

The second set of main experiments proved to contain the data needed to perform system identification. This set of experiments is intended to be used for validation, and for checking the reproducibility of the experiments.

Id.	File name	Mean load (J/cycle)	CR	PWM (%)	Mean temp.	Speed	CA50 mean	Comments
22	0306271113	240	21	97	174	2018	4.1	Temp step
23	0306271119	240 ±20	20 ±0.15	97.4 ±0.25	189	2020 ±200	3.4	Concurrent excitation
24	0306271123	310 ±20	20 ±0.15	96.6 ±0.25	160	2011 ±200	6.4	Concurrent excitation
25	0306271129	250 ±20	19.7 ±0.15	96.8 ±0.25	189	3030 ±200	5.7	Concurrent excitation
26	0306271134	300 ±20	18.6 ±0.1	96.8 ±0.2	196	3037 ±150	5.7	Concurrent excitation
27	0306271142	240 ±20	18.0 ±0.1	96.9 ±0.2	204	4031 ±100	5.8	Concurrent excitation
28	0306271146	300 ±15	17.5 ±0.1	96.9 ±0.2	203	4018 ±100	5.9	Concurrent excitation
29	0306271154	240 ±10	16.8 ±0.1	97.0 ±0.2	210	5022 ±50	7.1	Concurrent excitation

B Matlab files

This Appendix contains all the Matlab files used to obtain the results in the text.

B.1 loadData.m

This file is used to extract relevant data from the data-files. In the data files all data is stored in vectors. In this file the data is rearranged into matrices with one column for each cylinder and one row for each engine cycle. At this point offsets and linear trends are removed from the data before it is returned.

```
% [CA50, CR, pwm, Temp, IMEP,Speed] = loadData(fileName)
function [CA50,CR,PWM,Temp,FuelHeat,Speed] = loadData(fileName)

load(fileName);
NbrOfCyl = 5;
CA50 = double(CA50);
Temp = double(TInlet);
CR = double(CR);
PWM = double(PWM);
FuelHeat = double(FuelHeat);
Speed = double(Speed);

[n, m] = size(CA50);
cols = n*m/NbrOfCyl;

CA50 = detrend(reshape(CA50, NbrOfCyl, cols)');
CR = detrend(reshape(CR, NbrOfCyl, cols)');
Temp = detrend(reshape(Temp, NbrOfCyl, cols)');
FuelHeat = detrend(reshape(FuelHeat, NbrOfCyl, cols)');
PWM = detrend(reshape(PWM, NbrOfCyl, cols)');
Speed = detrend(reshape(Speed,NbrOfCyl, cols)');
```

B.2 zizv.m

This file is used to store the data obtained from the *loadData* file in structs containing one iddata object for each cylinder. The names of the input and output signals are stored in the iddata objects.

```
% [zi,zv] = zizv(fileName)
% Returns two structs containing iddata objects containing input signals
% [CR, TInlet, FuelHeat, Speed] and output signal CA50
% for each of the five cylinders.

function [zi,zv] = zizv(fileName)

[CA50,CR,PWM,Temp,FuelHeat,Speed]= loadData(fileName);
for i = 1:5
    eval(['zi.cyl', int2str(i),'= iddata(CA50(1:500,',int2str(i),...
            '), [CR(1:500,',int2str(i),'),Temp(1:500,',int2str(i),...
            '), FuelHeat(1:500,',int2str(i),'),Speed(1:500,',...
            int2str(i),')],''OutputName'',''CA50'','
            '''InputName'',{''CR'',''TInlet'',''FuelHeat'',''Speed''});']);

    eval(['zv.cyl', int2str(i),'= iddata(CA50(501:1000,',int2str(i),...
            '), [CR(501:1000,',int2str(i),'),Temp(501:1000,',int2str(i),...
            '), FuelHeat(501:1000,',int2str(i),'),Speed(501:1000,',...
            int2str(i),')],''OutputName'',''CA50'',''...
            '''InputName'',{''CR'',''TInlet'',''FuelHeat'',''Speed''});']);
```

```
end
```

B.3 complexity_test.m

```
% Collect the data from set 3
[zi1 zv1] = zizv('0306241002_Status.mat');
[zi2 zv2] = zizv('0306241011_Status.mat');
[zi3 zv3] = zizv('0306241145_Status.mat');
[zi4 zv4] = zizv('0306241150_Status.mat');
[zi5 zv5] = zizv('0306241154_Status.mat');
[zi6 zv6] = zizv('0306241200_Status.mat');
everything = merge(zi1.cyl1,zi1.cyl2,zi1.cyl3,zi1.cyl4,zi1.cyl5,...
                   zi2.cyl1,zi2.cyl2,zi2.cyl3,zi2.cyl4,zi2.cyl5,...
                   zi3.cyl1,zi3.cyl2,zi3.cyl3,zi3.cyl4,zi3.cyl5,...
                   zi4.cyl1,zi4.cyl2,zi4.cyl3,zi4.cyl4,zi4.cyl5,...
                   zi5.cyl1,zi5.cyl2,zi5.cyl3,zi5.cyl4,zi5.cyl5,...
                   zi6.cyl1,zi6.cyl2,zi6.cyl3,zi6.cyl4,zi6.cyl5);

for i = 1:15
 eval(['merged_1{',int2str(i),'}=n4sid(everything,',int2str(i),...
            ',''trace'',''on'',''DisturbanceModel'',''None'',''nk''',...
            ',[0,0,0,0]);']);
end

% Collect the data from set four
[zi11 zv11] = zizv('0306271119_Status.mat');
[zi22 zv22] = zizv('0306271123_Status.mat');
[zi33 zv33] = zizv('0306271129_Status.mat');
[zi44 zv44] = zizv('0306271134_Status.mat');
[zi55 zv55] = zizv('0306271142_Status.mat');
[zi66 zv66] = zizv('0306271146_Status.mat');
[zi77 zv77] = zizv('0306271154_Status.mat');
everything = merge(zi11.cyl1,zi11.cyl2,zi11.cyl3,zi11.cyl4,zi11.cyl5,...
                   zi22.cyl1,zi22.cyl2,zi22.cyl3,zi22.cyl4,zi22.cyl5,...
                   zi33.cyl1,zi33.cyl2,zi33.cyl3,zi33.cyl4,zi33.cyl5,...
                   zi44.cyl1,zi44.cyl2,zi44.cyl3,zi44.cyl4,zi44.cyl5,...
                   zi55.cyl1,zi55.cyl2,zi55.cyl3,zi55.cyl4,zi55.cyl5,...
                   zi66.cyl1,zi66.cyl2,zi66.cyl3,zi66.cyl4,zi66.cyl5,...
                   zi77.cyl1,zi77.cyl2,zi77.cyl3,zi77.cyl4,zi77.cyl5);
for i = 1:15

 eval(['merged_2{',int2str(i),'}=n4sid(everything,',int2str(i),...
            ',''trace'',''on'',''DisturbanceModel'',''None'',''nk''',...
            ',[0,0,0,0]);']);
end

for i = 1:15
    eval(['[yh, Trash] = compare(zi11.cyl3,merged_1{',int2str(i),...
            '},merged_2{',int2str(i),'});']);
    temp = corrcoef(zi11.cyl3.y,yh{1}.y);
    corr1(i,1) = temp(2,1);
    temp = corrcoef(zi11.cyl3.y,yh{2}.y);
    corr1(i,2) = temp(2,1);
    eval(['[yh, Trash] = compare(zi22.cyl3,merged_1{',int2str(i),...
            '},merged_2{',int2str(i),'});']);
    temp = corrcoef(zi22.cyl3.y,yh{1}.y);
    corr2(i,1) = temp(2,1);
    temp = corrcoef(zi22.cyl3.y,yh{2}.y);
    corr2(i,2) = temp(2,1);
    eval(['[yh, Trash] = compare(zi33.cyl3,merged_1{',int2str(i),...
            '},merged_2{',int2str(i),'});']);
```

```
    temp = corrcoef(zi33.cyl3.y,yh{1}.y);
    corr3(i,1) = temp(2,1);
    temp = corrcoef(zi33.cyl3.y,yh{2}.y);
    corr3(i,2) = temp(2,1);
    eval(['[yh, Trash] = compare(zi44.cyl3,merged_1{',int2str(i),...
            '},merged_2{',int2str(i),'});']);
    temp = corrcoef(zi44.cyl3.y,yh{1}.y);
    corr4(i,1) = temp(2,1);
    temp = corrcoef(zi44.cyl3.y,yh{2}.y);
    corr4(i,2) = temp(2,1);
    eval(['[yh, Trash] = compare(zi55.cyl3,merged_1{',int2str(i),...
            '},merged_2{',int2str(i),'});']);
    temp = corrcoef(zi55.cyl3.y,yh{1}.y);
    corr5(i,1) = temp(2,1);
    temp = corrcoef(zi55.cyl3.y,yh{2}.y);
    corr5(i,2) = temp(2,1);
    eval(['[yh, Trash] = compare(zi66.cyl3,merged_1{',int2str(i),...
            '},merged_2{',int2str(i),'});']);
    temp = corrcoef(zi66.cyl3.y,yh{1}.y);
    corr6(i,1) = temp(2,1);
    temp = corrcoef(zi66.cyl3.y,yh{2}.y);
    corr6(i,2) = temp(2,1);
    eval(['[yh, Trash] = compare(zi77.cyl3,merged_1{',int2str(i),...
            '},merged_2{',int2str(i),'});']);
    temp = corrcoef(zi77.cyl3.y,yh{1}.y);
    corr7(i,1) = temp(2,1);
    temp = corrcoef(zi77.cyl3.y,yh{2}.y);
    corr7(i,2) = temp(2,1);
end

mean_corr = (corr1 + corr2 + corr3 + corr4 + corr5 + corr6 + corr7)'/7;
min_corr = [min([corr1(:,1)';corr2(:,1)';corr3(:,1)';corr4(:,1)';...
            corr5(:,1)';corr6(:,1)';corr7(:,1)']);min([corr1(:,2)';...
            corr2(:,2)';corr3(:,2)';corr4(:,2)';corr5(:,2)';...
            corr6(:,2)';corr7(:,2)'])];
max_corr = [max([corr1(:,1)';corr2(:,1)';corr3(:,1)';corr4(:,1)';...
            corr5(:,1)';corr6(:,1)';corr7(:,1)']);max([corr1(:,2)';...
            corr2(:,2)';corr3(:,2)';corr4(:,2)';corr5(:,2)';...
            corr6(:,2)';corr7(:,2)'])];

figure(1);
plot(max_corr(1,:),'r-x');
hold on;
grid on;
plot(mean_corr(1,:),'b-');
plot(min_corr(1,:),'g-+');
legend('Max correlation','Mean correlation','Min correlation',0);
hold off;

figure(2);
plot(max_corr(2,:),'r-x');
hold on;
grid on;
plot(mean_corr(2,:),'b-');
plot(min_corr(2,:),'g-+');
legend('Max correlation','Mean correlation','Min correlation',0);
hold off;
```

B.4 single_many_test.m

```
[zi1 zv1] = zizv('0306241002_Status.mat');
```

```
[zi2 zv2] = zizv('0306241011_Status.mat');
[zi3 zv3] = zizv('0306241145_Status.mat');
[zi4 zv4] = zizv('0306241150_Status.mat');
[zi5 zv5] = zizv('0306241154_Status.mat');
[zi6 zv6] = zizv('0306241200_Status.mat');
[zi7 zv7] = zizv('0306271119_Status.mat');
[zi8 zv8] = zizv('0306271123_Status.mat');
[zi9 zv9] = zizv('0306271129_Status.mat');
[zi10 zv10] = zizv('0306271134_Status.mat');
[zi11 zv11] = zizv('0306271142_Status.mat');
[zi12 zv12] = zizv('0306271146_Status.mat');
[zi13 zv13] = zizv('0306271154_Status.mat');
everything = merge(zi1.cyl1,zi1.cyl2,zi1.cyl3,zi1.cyl4,zi1.cyl5,...
                   zi2.cyl1,zi2.cyl2,zi2.cyl3,zi2.cyl4,zi2.cyl5,...
                   zi3.cyl1,zi3.cyl2,zi3.cyl3,zi3.cyl4,zi3.cyl5,...
                   zi4.cyl1,zi4.cyl2,zi4.cyl3,zi4.cyl4,zi4.cyl5,...
                   zi5.cyl1,zi5.cyl2,zi5.cyl3,zi5.cyl4,zi5.cyl5,...
                   zi6.cyl1,zi6.cyl2,zi6.cyl3,zi6.cyl4,zi6.cyl5,...
                   zi7.cyl1,zi7.cyl2,zi7.cyl3,zi7.cyl4,zi7.cyl5,...
                   zi8.cyl1,zi8.cyl2,zi8.cyl3,zi8.cyl4,zi8.cyl5,...
                   zi9.cyl1,zi9.cyl2,zi9.cyl3,zi9.cyl4,zi9.cyl5,...
                   zi10.cyl1,zi10.cyl2,zi10.cyl3,zi10.cyl4,zi10.cyl5,...
                   zi11.cyl1,zi11.cyl2,zi11.cyl3,zi11.cyl4,zi11.cyl5,...
                   zi12.cyl1,zi12.cyl2,zi12.cyl3,zi12.cyl4,zi12.cyl5,...
                   zi13.cyl1,zi13.cyl2,zi13.cyl3,zi13.cyl4,zi13.cyl5);
zi1 = merge(zi1.cyl1,zi1.cyl2,zi1.cyl3,zi1.cyl4,zi1.cyl5);
zi2 = merge(zi2.cyl1,zi2.cyl2,zi2.cyl3,zi2.cyl4,zi2.cyl5);
zi3 = merge(zi3.cyl1,zi3.cyl2,zi3.cyl3,zi3.cyl4,zi3.cyl5);
zi4 = merge(zi4.cyl1,zi4.cyl2,zi4.cyl3,zi4.cyl4,zi4.cyl5);
zi5 = merge(zi5.cyl1,zi5.cyl2,zi5.cyl3,zi5.cyl4,zi5.cyl5);
zi6 = merge(zi6.cyl1,zi6.cyl2,zi6.cyl3,zi6.cyl4,zi6.cyl5);
zi7 = merge(zi7.cyl1,zi7.cyl2,zi7.cyl3,zi7.cyl4,zi7.cyl5);
zi8 = merge(zi8.cyl1,zi8.cyl2,zi8.cyl3,zi8.cyl4,zi8.cyl5);
zi9 = merge(zi9.cyl1,zi9.cyl2,zi9.cyl3,zi9.cyl4,zi9.cyl5);
zi10 = merge(zi10.cyl1,zi10.cyl2,zi10.cyl3,zi10.cyl4,zi10.cyl5);
zv11 = zi11.cyl1;
zi11 = merge(zi11.cyl1,zi11.cyl2,zi11.cyl3,zi11.cyl4,zi11.cyl5);
zv12 = zi12.cyl1
zi12 = merge(zi12.cyl1,zi12.cyl2,zi12.cyl3,zi12.cyl4,zi12.cyl5);
zi13 = merge(zi13.cyl1,zi13.cyl2,zi13.cyl3,zi13.cyl4,zi13.cyl5);

m1 = n4sid(zi1, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m2 = n4sid(zi2, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m3 = n4sid(zi3, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m4 = n4sid(zi4, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m5 = n4sid(zi5, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m6 = n4sid(zi6, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m7 = n4sid(zi7, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m8 = n4sid(zi8, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m9 = n4sid(zi9, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m10 = n4sid(zi10, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
```

```
m11 = n4sid(zi11, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m12 = n4sid(zi12, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
m13 = n4sid(zi13, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
merged = n4sid(everything, 3, 'trace', 'off', 'DisturbanceModel',
'None','nk',[0,0,0,0]);
correlations = zeros(2,13);
[yh, Trash] = compare(zv1.cyl3,m1,merged);
c1 = corrcoef(zv1.cyl3.y,yh{1}.y);
c2 = corrcoef(zv1.cyl3.y,yh{2}.y);
correlations(1,1) = c1(1,2);
correlations(2,1) = c2(1,2);
[yh, Trash] = compare(zv2.cyl3,m2,merged);
c1 = corrcoef(zv2.cyl3.y,yh{1}.y);
c2 = corrcoef(zv2.cyl3.y,yh{2}.y);
correlations(1,2) = c1(1,2);
correlations(2,2) = c2(1,2);
[yh, Trash] = compare(zv3.cyl3,m3,merged);
c1 = corrcoef(zv3.cyl3.y,yh{1}.y);
c2 = corrcoef(zv3.cyl3.y,yh{2}.y);
correlations(1,3) = c1(1,2);
correlations(2,3) = c2(1,2);
[yh, Trash] = compare(zv4.cyl3,m4,merged);
c1 = corrcoef(zv4.cyl3.y,yh{1}.y);
c2 = corrcoef(zv4.cyl3.y,yh{2}.y);
correlations(1,4) = c1(1,2);
correlations(2,4) = c2(1,2);
[yh, Trash] = compare(zv5.cyl3,m5,merged);
c1 = corrcoef(zv5.cyl3.y,yh{1}.y);
c2 = corrcoef(zv5.cyl3.y,yh{2}.y);
correlations(1,5) = c1(1,2);
correlations(2,5) = c2(1,2);
[yh, Trash] = compare(zv6.cyl3,m6,merged);
c1 = corrcoef(zv6.cyl3.y,yh{1}.y);
c2 = corrcoef(zv6.cyl3.y,yh{2}.y);
correlations(1,6) = c1(1,2);
correlations(2,6) = c2(1,2);
[yh, Trash] = compare(zv7.cyl3,m7,merged);
c1 = corrcoef(zv7.cyl3.y,yh{1}.y);
c2 = corrcoef(zv7.cyl3.y,yh{2}.y);
correlations(1,7) = c1(1,2);
correlations(2,7) = c2(1,2);
[yh, Trash] = compare(zv8.cyl3,m8,merged);
c1 = corrcoef(zv8.cyl3.y,yh{1}.y);
c2 = corrcoef(zv8.cyl3.y,yh{2}.y);
correlations(1,8) = c1(1,2);
correlations(2,8) = c2(1,2);
[yh, Trash] = compare(zv9.cyl3,m9,merged);
c1 = corrcoef(zv9.cyl3.y,yh{1}.y);
c2 = corrcoef(zv9.cyl3.y,yh{2}.y);
correlations(1,9) = c1(1,2);
correlations(2,9) = c2(1,2);
[yh, Trash] = compare(zv10.cyl3,m10,merged);
c1 = corrcoef(zv10.cyl3.y,yh{1}.y);
c2 = corrcoef(zv10.cyl3.y,yh{2}.y);
correlations(1,10) = c1(1,2);
correlations(2,10) = c2(1,2);
[yh, Trash] = compare(zv11,m11,merged);
c1 = corrcoef(zv11.y,yh{1}.y);
```

```
c2 = corrcoef(zv11.y,yh{2}.y);

correlations(1,11) = c1(1,2);
correlations(2,11) = c2(1,2);
[yh, Trash] = compare(zv12,m12,merged);
c1 = corrcoef(zv12.y,yh{1}.y);
c2 = corrcoef(zv12.y,yh{2}.y);
correlations(1,12) = c1(1,2);
correlations(2,12) = c2(1,2);
[yh, Trash] = compare(zv13.cyl3,m13,merged);
c1 = corrcoef(zv13.cyl3.y,yh{1}.y);
c2 = corrcoef(zv13.cyl3.y,yh{2}.y);
correlations(1,13) = c1(1,2);
correlations(2,13) = c2(1,2);

hold off;
plot(correlations(1,:),'r-');
hold on;
plot(correlations(2,:),'b-+');
legend('operating point individual model','Merged model');
title('Comparing cylinder individual models to merged model');
xlabel('Operating point');
ylabel('Correlation');
grid on;
```

B.5 residual_analysis.m

```
[zi,zv] = zizv('0306271123_Status.mat');
if ~exist('model')
    create_model;
end
simulated = zeros(500,5);
simulated(:,1) = idsim(zv.cyl1.u,model);
simulated(:,2) = idsim(zv.cyl2.u,model);
simulated(:,3) = idsim(zv.cyl3.u,model);
simulated(:,4) = idsim(zv.cyl4.u,model);
simulated(:,5) = idsim(zv.cyl5.u,model);

residuals = zeros(500,5);
residuals(:,1) = zv.cyl1.y-simulated(:,1);
residuals(:,2) = zv.cyl2.y-simulated(:,2);
residuals(:,3) = zv.cyl3.y-simulated(:,3);
residuals(:,4) = zv.cyl4.y-simulated(:,4);
residuals(:,5) = zv.cyl5.y-simulated(:,5);
figure(1);
mean_res = mean(residuals')';
plot(mean_res,'b-');
title('Mean residuals');
xlabel('Engine cycles');
ylabel('Degrees ATDC');

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

figure(3);
hold off;
plot(simulated(:,2),'b');
```

```
hold on;
plot(zv.cyl2.y,'r');
grid on;
title('Measured and simulated output');
xlabel('Engine cycles');
ylabel('Degrees ATDC');
legend('Simulated output','Measured output',0);
```

B.6 create_model.m

This is the file that creates a model from all data contained in the files indexed 15-21 and 23-28. All data in the mentioned files is first merged into one big iddata object, this object is then used as a parameter to the function n4sid which creates the model.

```
[zi1 trash] = zizv('0306241002_Status.mat');
[zi2 trash] = zizv('0306241011_Status.mat');
[zi3 trash] = zizv('0306241145_Status.mat');
[zi4 trash] = zizv('0306241150_Status.mat');
[zi5 trash] = zizv('0306241154_Status.mat');
[zi6 trash] = zizv('0306241200_Status.mat');
[zi7 trash] = zizv('0306271119_Status.mat');
[zi8 trash] = zizv('0306271123_Status.mat');
[zi9 trash] = zizv('0306271129_Status.mat');
[zi10 trash] = zizv('0306271134_Status.mat');
[zi11 trash] = zizv('0306271142_Status.mat');
[zi12 trash] = zizv('0306271146_Status.mat');
[zi13 trash] = zizv('0306271154_Status.mat');
everything = merge(zi1.cyl1,zi1.cyl2,zi1.cyl3,zi1.cyl4,zi1.cyl5,...
                   zi2.cyl1,zi2.cyl2,zi2.cyl3,zi2.cyl4,zi2.cyl5,...
                   zi3.cyl1,zi3.cyl2,zi3.cyl3,zi3.cyl4,zi3.cyl5,...
                   zi4.cyl1,zi4.cyl2,zi4.cyl3,zi4.cyl4,zi4.cyl5,...
                   zi5.cyl1,zi5.cyl2,zi5.cyl3,zi5.cyl4,zi5.cyl5,...
                   zi6.cyl1,zi6.cyl2,zi6.cyl3,zi6.cyl4,zi6.cyl5,...
                   zi7.cyl1,zi7.cyl2,zi7.cyl3,zi7.cyl4,zi7.cyl5,...
                   zi8.cyl1,zi8.cyl2,zi8.cyl3,zi8.cyl4,zi8.cyl5,...
                   zi9.cyl1,zi9.cyl2,zi9.cyl3,zi9.cyl4,zi9.cyl5,...
                   zi10.cyl1,zi10.cyl2,zi10.cyl3,zi10.cyl4,zi10.cyl5,...
                   zi11.cyl1,zi11.cyl2,zi11.cyl3,zi11.cyl4,zi11.cyl5,...
                   zi12.cyl1,zi12.cyl2,zi12.cyl3,zi12.cyl4,zi12.cyl5,...
                   zi13.cyl1,zi13.cyl2,zi13.cyl3,zi13.cyl4,zi13.cyl5);

model = n4sid(everything, 3, 'trace', 'off', 'DisturbanceModel',...
        'None','nk',[0,0,0,0]);
```

B.7 observability_controllability_test.m

```
if ~exist('model')
    create_model;
end
W_o = obsv(model.A,model.C)
if rank(W_o) == size(model.A,1)
    disp('Model is observable');
else
    disp('Model is NOT observable');
end

W_c = ctrb(model.A,model.B)
if rank(W_c) == size(model.A,1)
    disp('Model is controllable');
else
    disp('Model is not controllable');
```

```
end
```

B.8 create_sensitivity_function.m

```
if ~exist('model')
    create_model;
end
if ~exist('RLQG')
    create_kalman;
end
plant_tf = tf(ss(model.A,model.B,model.C,model.D,1,'InputName',...
    {'CR','TInlet','FuelHeat','Speed'},'OutputName','CA50'));
regulator_tf = tf([1],[1,-1],1) * tf(ss(RLQG.A,RLQG.B,RLQG.C,RLQG.D,1,...
    'InputName','CA50','OutputName',{'CR','TInlet','FuelHeat','Speed'}));
L = plant_tf * regulator_tf;
S = (1 - L)^-1
T = 1 - S
figure(1);
bode(S);
grid on;
title('Sensitivity function');
figure(2);
bode(T);
grid on;
title('Complementary sensitivity function');
```

B.9 HCCI.m

```
function [sys,x0,str,ts] = HCCI(t,x,u,flag,A,B,C,D,w_power,e_power)
%HCCI(t,x,u,flag,A,B,C,D,w_power,e_power)
%
%   Parameters A, B, C and D are the matrices of the discrete state space
%   representation of the engine.
%   w_power and e_power are the variances of the noise.
%   x(t+1) = A*x(t) + B*u(t) + w(t)
%   y(t)   = C*x(t) + D*u(t) + e(t)

switch flag,

  % Initialization %
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C);

  % Derivatives % Not used here
  case 1,
    sys=mdlDerivatives(t,x,u);

  % Update %
  case 2,
    sys=mdlUpdate(t,x,u,A,B,w_power);

  % Outputs %
  case 3,
    sys=mdlOutputs(t,x,u,C,D,e_power);

  % GetTimeOfNextVarHit %
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate %
```

```
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags %
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
% end HCCI

%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C)
 sizes = simsizes;
 sizes.NumContStates  = 0;
 sizes.NumDiscStates  = size(A,1);
 sizes.NumOutputs     = size(C,1);
 sizes.NumInputs      = size(B,2);
 sizes.DirFeedthrough = 1;
 sizes.NumSampleTimes = 1;   % at least one sample time is needed

 sys = simsizes(sizes);
 % initialize the initial conditions
 x0  = zeros(1,size(A,1));
 % str is always an empty matrix
 str = [];
 % initialize the array of sample times
 ts  = [1 0];
% end mdlInitializeSizes

%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
function sys=mdlDerivatives(t,x,u)
    sys = [];
% end mdlDerivatives

%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
function sys=mdlUpdate(t,x,u,A,B,w_power)
    w = zeros(length(w_power),1);
    for i = 1:length(w)
        w(i) = randn*sqrt(w_power(i));
    end
    sys = A*x+B*u+w;
% end mdlUpdate

%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
function sys=mdlOutputs(t,x,u,C,D,e_power)
    e = randn*sqrt(e_power);
    sys = C*x + D*u + e;
% end mdlOutputs
```

```
%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
    sampleTime = 1;    %  Example, set the next hit to be one second later.
    sys = t + sampleTime;
% end mdlGetTimeOfNextVarHit

%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
function sys=mdlTerminate(t,x,u)
    sys = [];
% end mdlTerminate
```

B.10 create_lqg.m

```
if ~exist('model');
    create_model; % Creates a model of the engine.
end
weights; % Initializes the Q and R matrices.

[kest,L,P] = kalman(ss(model.A,[model.B,eye(3)],model.C,[model.D,0,0,1]...
    ,1,'InputName',{'CR','TInlet','FuelHeat','Speed','noise_1','noise_2'...
    ,'noise_3'},'OutputName','CA50'),diag(process_noise),measurement_noise);

[K,S,E] = DLQR(model.A,[model.B(:,1:2),zeros(3,2)],Q,R);
RLQG = LQGREG(kest,K,'current');
```

B.11 lqg_regulator.m

```
function [sys,x0,str,ts] = lqg_regulator(t,x,u,flag,...
    A,B,C,D,w_power,e_power,Q,R,N)
% lqg_regulator(t,x,u,flag,A,B,C,D,w_power,e_power,Q,R,N)
%      Creates and acts as a LQG regulator for the process described by the
%      matrices A, B, C and D and the noise variances w_power and e_power.
%      The process is described by the following equations
%
%          x(t+1) = A*x(t) + B*u(t) + w(t)
%          y(t) = C*x(t) + D*u(t) + e(t)
%
%      The matrices Q, R and N are the weighting matrices of the loss
%      function
%
%          sum(x' Q x + u' R u + u' N x)

switch flag,

  % Initialization %
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D,w_power,e_power,Q,R,N);

  % Derivatives %
  case 1,
    sys=mdlDerivatives(t,x,u);
```

```
  % Update %
  case 2,
    sys=mdlUpdate(t,x,u);

  % Outputs %
  case 3,
    sys=mdlOutputs(t,x,u);

  % GetTimeOfNextVarHit %
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  % Terminate %
  case 9,
    sys=mdlTerminate(t,x,u);

  % Unexpected flags %
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);

end
% end lqg_regulator

%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
function [sys,x0,str,ts]=mdlInitializeSizes(A,B,C,D,w_power,e_power,Q,R,N)
 sizes = simsizes;
 RLQG = createLQGRegulator(A,B,C,D,w_power,e_power,Q,R,N)

 sizes.NumContStates  = 0;
 sizes.NumDiscStates  = size(RLQG.A,2) + prod(size(RLQG.A)) +...
        prod(size(RLQG.B)) + prod(size(RLQG.C)) + prod(size(RLQG.D)) + 8;
 sizes.NumOutputs     = 2;
 sizes.NumInputs      = 1;
 sizes.DirFeedthrough = 1;
 sizes.NumSampleTimes = 1;

 sys = simsizes(sizes);

 x0  = zeros(sizes.NumDiscStates,1);
 currentIndex = size(RLQG.A,2)+1; % Reserve space for the states.
 % The regulator needs to be carried as states
 for i = 1:size(RLQG.A,1)
        x0(currentIndex:currentIndex+size(RLQG.A,2)-1) = RLQG.A(i,:);
        currentIndex = currentIndex+size(RLQG.A,2);
 end

 for i = 1:size(RLQG.B,1)
        x0(currentIndex:currentIndex+size(RLQG.B,2)-1) = RLQG.B(i,:);
        currentIndex = currentIndex+size(RLQG.B,2);
 end

 for i = 1:size(RLQG.C,1)
        x0(currentIndex:currentIndex+size(RLQG.C,2)-1) = RLQG.C(i,:);
        currentIndex = currentIndex+size(RLQG.C,2);
 end

 for i = 1:size(RLQG.D,1)
        x0(currentIndex:currentIndex+size(RLQG.D,2)-1) = RLQG.D(i,:);
```

```
        currentIndex = currentIndex+size(RLQG.D,2);
 end
 % Finally add the sizes of the matrices
 x0(currentIndex:currentIndex+7) =
[size(RLQG.A),size(RLQG.B),size(RLQG.C),size(RLQG.D)];

 % str is always an empty matrix
 str = [];

 % initialize the array of sample times
 ts  = [1 0];
% end mdlInitializeSizes

%=============================================================================
% mdlDerivatives
% Return the derivatives for the continuous states.
%=============================================================================
function sys=mdlDerivatives(t,x,u)
    sys = [];
% end mdlDerivatives

%=============================================================================
% mdlUpdate
% Handle discrete state updates, sample time hits, and major time step
% requirements.
%=============================================================================
function sys=mdlUpdate(t,x,u)
 [A,B,C,D] = getMatrices(x);
    sys(1:3) = A * x(1:3) + B * u;
    sys(4:size(x,1)) = x(4:size(x,1));
% end mdlUpdate

%=============================================================================
% mdlOutputs
% Return the block outputs.
%=============================================================================
function sys=mdlOutputs(t,x,u)
 [A,B,C,D] = getMatrices(x);
 sys = C * x(1:3) + D * u;
    sys = saturate(sys);
% end mdlOutputs

%=============================================================================
% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block.  Note that the result is
% absolute time.  Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.
%=============================================================================
function sys=mdlGetTimeOfNextVarHit(t,x,u)
 sampleTime = 1;
 sys = t + sampleTime;
% end mdlGetTimeOfNextVarHit

%=============================================================================
% mdlTerminate
% Perform any end of simulation tasks.
%=============================================================================
function sys=mdlTerminate(t,x,u)
 sys = [];
% end mdlTerminate
```

```
%=============================================================================
% Creates the kalman filter
%=============================================================================
function RLQG = createLQGRegulator(A,B,C,D,w_power,e_power,Q,R,N)
 Q1 = diag(w_power);
 R1 = e_power;
    InputName = {'CR','TInlet','FuelHeat','Speed'};
    for i = 1:length(w_power)
        InputName{length(InputName) + 1} = ['noise_',int2str(i)];
    end
 [kest,L,P] = kalman(ss(A,[B,eye(size(A,1))],C,[D,0,0,1],1,...
        'InputName', InputName,'OutputName','CA50'),Q1,R1);
    % The last two process inputs are removed since these are not
    % accessible to the regulator.
    [K,S,E] = DLQR(A,[B(:,1:2),zeros(3,2)],Q,R,N);
    RLQG = LQGREG(kest,K,'current');
% end createKalman

function [A,B,C,D] = getMatrices(x);
    % Start by obtaining the matrix sizes
    currentIndex = length(x)-7;

    size_A = x(currentIndex:currentIndex+1);
    currentIndex = currentIndex + 2;
    size_B = x(currentIndex:currentIndex+1);
    currentIndex = currentIndex + 2;
    size_C = x(currentIndex:currentIndex+1);
    currentIndex = currentIndex + 2;
    size_D = x(currentIndex:currentIndex+1);
    A = zeros(size_A');
    B = zeros(size_B');
    C = zeros(size_C');
    D = zeros(size_D');

    currentIndex = size_A(1) + 1; % This is where the matrices begin
    for i = 1:size_A(1)
        A(i,:) = x(currentIndex:currentIndex+size_A(2)-1)';
        currentIndex = currentIndex+size_A(2);
    end
    for i = 1:size_B(1)
        B(i,:) = x(currentIndex:currentIndex+size_B(2)-1)';
        currentIndex = currentIndex+size_B(2);
    end
    for i = 1:size_C(1)
        C(i,:) = x(currentIndex:currentIndex+size_C(2)-1)';
        currentIndex = currentIndex+size_C(2);
    end
    for i = 1:size_D(1)
        D(i,:) = x(currentIndex:currentIndex+size_D(2)-1)';
        currentIndex = currentIndex+size_D(2);
    end
% end getMatrices

function sys = saturate(s)
    % Saturate CR
    sys(1) = max(s(1),-9);
    sys(1) = min(s(1),3);
    % Saturate IAT
    sys(2) = max(s(2),-100);
    sys(2) = min(s(2),100);
```

```
% end saturate
```

B.12 test_noise.m

```
function test_noise
    load('0306121008_Status.mat');
    CA50 = reshape(CA50, 5, 1000)';
    noise = detrend(CA50(400:1000,1)');
    figure(1)
    plot(400:1000,noise);
    grid on;
    figure(2)
    [P,F] = spectrum(noise, 256, 100, [], 1);
    plot(F, P(:, 1));
    var(noise)
```

B.13 create_regulator.m

```
[kest,L,P] =
kalman(ss(model.A,[model.B,eye(3)],model.C,[model.D,0,0,1],1,...

'InputName',{'CR','TInlet','FuelHeat','Speed','noise_1','noise_2','noise_3'}
,'OutputName','CA50')...
    ,diag(regulator_process_noise),regulator_measurement_noise);

[K,S,E] = DLQR(model.A,[model.B(:,1:2),zeros(3,2)],Q,R);
RLQG = LQGREG(kest,K,'current');

A = RLQG.A;
B = RLQG.B;
C = RLQG.C;
D = RLQG.D;

z = zeros(size(A));
regulator_A = [A,z,z,z,z;z,A,z,z,z;z,z,A,z,z;z,z,z,A,z;z,z,z,z,A];
regulator_B = [B,zeros(3,4);...
               zeros(3,1),B,zeros(3,3);...
               zeros(3,2),B,zeros(3,2);...
               zeros(3,3),B,zeros(3,1);...
               zeros(3,4),B];
regulator_C = [C(1,:)/5,C(1,:)/5,C(1,:)/5,C(1,:)/5,C(1,:)/5;...
               C(2,:),zeros(1,12);...
               zeros(1,3),C(2,:),zeros(1,9);...
               zeros(1,6),C(2,:),zeros(1,6);...
               zeros(1,9),C(2,:),zeros(1,3);...
               zeros(1,12),C(2,:)];
regulator_D =
[D(1)/5,D(1)/5,D(1)/5,D(1)/5,D(1)/5;D(2),0,0,0,0;0,D(2),0,0,0;...
        0,0,D(2),0,0;0,0,0,D(2),0;0,0,0,0,D(2)];

clear A B C D kest L P z K S E RLQG;
```

B.14 LQG_Controller.java

```
/**
 * Regulator for ignition phasing control of the five cylinder
 * variable CR HCCI Engine.
 * It is assumed that the controller has six outputs. These are
 * CR (for the five cylinders in parallel and IAT (individual
 * control for each cylinder). CR is controlled for all cylinders
 * in paralell.
 * The number of inputs to the controller is arbitrary, and
 * so is the complexity of the controller.
```

```
 */
public class LQG_Controller {

  // Regulator matrices
  private double[][] A, B, C, D, x, oldInputs;
  private int index;
  private double CR;

  /**
   * Constructs a controller object.
   * The parameters are the matrices of the regulator for
   * one cylinder in state space form.
   */
  public LQG_Controller(double[][] A, double[][] B,
                        double[][] C, double[][] D) {
    this.A = A;
    this.B = B;
    this.C = C;
    this.D = D;
    index = 0;
    // Reserve space for the five latest inputs
    oldInputs = new double[5][B[0].length];
    // Reserve space for the states of all five cylinders
    x = new double[5][A.length];
  }

  /**
   * Updates the controller states for cylinder cylNbr.
   */
  public void updateStates(double[] inputs, int cylNbr) {
    double[] newX = new double[A.length];
    for (int i = 0; i < A.length; i++) {
      // Calculate the contribution from the A matrix
      for (int k = 0; k < A.length; k++) {
        newX[i] = newX[i] + A[i][k] * x[cylNbr][k];
      }
      // Calculate the contribution from the B matrix
      for (int k = 0; k < inputs.length; k++) {
        newX[i] = newX[i] + B[i][k] * inputs[k];
      }
    }
    // Update the states in the x-matrix
    for (int i = 0; i < A.length; i++) {
      x[cylNbr][i] = newX[i];
    }
  }

  /**
   * Calculates the CR output.
   * This function uses stored values of the inputs
   * to the different cylinders.
   */
  public double calculateCR() {
    double[] meanStates = new double[A.length];
    double[] meanInputs = new double[B[0].length];
    double output = 0;
    // Calculate the mean of the states
    for (int i = 0; i < meanStates.length; i++) {
      for (int k = 0; k < 5; k++) {
        meanStates[i] = meanStates[i] + x[k][i];
      }
```

```
      meanStates[i] = meanStates[i] * 0.2;
    }
    // Calculate the mean of the inputs
    for (int i = 0; i < meanInputs.length; i++) {
      for (int k = 0; k < 5; k++) {
        meanInputs[i] = meanInputs[i] + oldInputs[k][i];
      }
      meanInputs[i] = meanInputs[i] * 0.2;
    }
    // Calculate the contribution from the states
    for (int i = 0; i < meanStates.length; i++) {
      output = output + C[0][i] * meanStates[i];
    }
    // Calculate the contribution from the inputs
    for (int i = 0; i < meanInputs.length; i++) {
      output = output + D[0][i] * meanInputs[i];
    }
    return output;
  }

  /**
   * Calculates the IAT output.
   * This method also updates the stored input values
   */
  public double calculateIAT(double[] inputs,int cylNbr) {
    double output = 0;
    // Calculate the contribution from the states
    for (int i = 0; i < x[0].length; i++) {
      output = output + C[1][i] * x[cylNbr][i];
    }
    // Calculate the contribution from the inputs
    // and update the stored values
    for (int i = 0; i < inputs.length; i++) {
      output = output + C[1][i] * inputs[i];
      oldInputs[cylNbr][i] = inputs[i];
    }
    return output;
  }

  /**
   * Calculates the controller output after each cylinder
   * cycle. IAT is updated at every call of this method.
   * CR is update once per engine cycle (after cylinder
   * five has compled a cycle).
   */
  public double[] cycleFinished(double[] inputs, int cylNbr) {
    double IAT = calculateIAT(inputs, cylNbr);
    if (cylNbr == 5) {
      CR = calculateCR();
    }
    return new double[]{CR,IAT};
  }
}
```