Cylinder Individual Lambda Feedback Control in an SI Engine

Examensarbete utfört i Fordonssystem
vid Tekniska Högskolan i Linköping
av

Patrik Berggren
Andrej Perkovic

Reg nr: LiTH-ISY-EX-1649

Cylinder Individual Lambda Feedback Control in an SI Engine

Examensarbete utfört i Fordonssystem
vid Tekniska Högskolan i Linköping
av

Patrik Berggren
Andrej Perkovic

Reg nr: LiTH-ISY-EX-1649

Supervisor: **Mikael Glavenius, Mecel AB**
Tomas Henriksson, LiTH
Lars Nielsen, LiTH

Examiner: **Lars Nielsen**

Linköping, January 16, 1996.

Abstract

In order to meet future standards regarding exhaust pollutants from SI engines, precise control of the air fuel ratio is essential. To enable usage of fuel injectors with less accuracy and to compensate for differences between cylinders due to wear, a cylinder individual control of lambda is needed. In this thesis we develop a non model based multi sensor controller to accomplish the above mentioned tasks. The presented controller uses an absolute measurement of lambda at the confluence point and relative cylinder individual lambda measurements. This is done to prepare for an exchange to cylinder individual lambdas provided by an ionization current algorithm, that is being developed at Mecel AB. The results are promising and the controller performance, including speed and accuracy, is superior to the conventional lambda controller also described in the thesis.

Key Word: conventional lambda control, cylinder individual, multi sensor lambda control.

Acknowledgments

This project is performed in cooperation between Mecel AB and the Division of Vehicular Systems at Linköping University. We wish to thank our supervisors Mikael Glavenius at Mecel AB and Lars Nielsen and Tomas Henriksson at the Division of Vehicular Systems at Linköping University. Special thanks to Simon Edlund for computer assistance, Lars Eriksson for helping us with Selma, Mattias Nyberg for providing Figure 2.1 and Magnus Pettersson for general trouble shooting and LaTeX support.

Notation

Operators and Functions

$\div$	output=1/input

Symbols

α	throttle angle
λ	normalized air fuel ratio
λ_i	normalized air fuel ratio in cylinder i
p_{man}	inlet manifold absolute pressure
m_a	air mass
m_f	fuel mass
m_{ff}	fuel film mass
m_{fi}	injected fuel mass
m_{fv}	vaporized fuel mass
f_c	cut–off frequency
K, K_y, T_i, T_{iy}	controller parameters
e, e_1, e_2, e_3, e_4	difference between controller inputs
u, t_1, t_2, t_3, t_4	controller output signals
T_1, T_2	settling times
t_{del}	transport delay to the confluence point
t_{delcyl}	transport delay to close to cylinder sensor
t_{inj}	total fuel injection time (basic injection time + lambda correction time)
u_{p-p}	u peak to peak value

Abbreviations

SI	Spark Ignition
A/F	Air/Fuel
$(A/F)_s$	Air/Fuel at stoichiometry
UEGO	Universal Exhaust Gas Oxygen
EGO	Exhaust Gas Oxygen
LQG	Linear Quadratic Gaussian
rpm	revolutions per minute
TWC	Three–Way Catalyst
PID	Proportional–Integral–Derivative
cyl	cylinder
cp	confluence point
sc	single cylinder
rms	root mean square
ref	reference value
bhp	brake horse–power

Contents

1 Introduction

1.1 Problem Specification and Objective

The objective is to find a control structure that can control lambda in a cylinder individual way. This shall be done in three stages:

- Cylinder individual lambda control using a UEGO sensor at the confluence point and UEGO sensors located close[1] to each cylinder.
- As above but the cylinder by cylinder placed sensors have an unknown offset.
- Cylinder individual lambda control using a UEGO sensor at the confluence point and cylinder individual lambdas estimated by an ionization current algorithm developed by Mecel AB.

The possibility to accomplish the last stage depends on the complexity of the control problem and the work done by Mecel AB in developing an ionization current algorithm.

To get some results to compare the cylinder individual controller with, the following stages shall be performed first:

- Conventional lambda control using an EGO sensor at the confluence point.
- As above but with a UEGO sensor.

1.2 Limitations

Below are the most important limitations listed.

- Locked fuel map and ignition angle during experiments.
- No exhaust gas recirculation (EGR).
- Control at three operating points, see Section 3.1.
- Lambda correction time limited to ± 2 ms.
- Not a perfectly tuned throttle control.
- All controllers use a reference value corresponding to $\lambda = 1$ in order to simplify comparisons between different control structures.
- A limited sample rate of about 2000 Hz in the real time system.
- A resolution of 4.88 mV in the D/A–card corresponding to a 0.5 % error in lambda for worst case.
- Time based sampling instead of event based.

[1] About 12 cm downstream the exhaust manifold.

1.3 Methods

Controlling lambda includes a lot of difficulties to deal with. For example, fuel injections taking place at discrete moments, non linearities in sensors and engine dynamics, systems (cylinders) running parallel to each other, lambda control gives only a limited correction to the basic injection time and cycle to cycle variations. Note that we can't measure the A/F ratio in the mixture entering the cylinders, which we want to control. We can only measure the oxygen content in the exhaust gases and by this gain information about the A/F ratio.

To simplify the work we only use standard PI controllers. The reason for not using PID controllers is that there is a high noise level in the laboratory and derivating a noisy signal causes problems. Recall that more sophisticated methods like LQG–design always contain P and I parts as basic blocks. This shows that studying PI control is a good first approach. Remember: try simple things first.

1.4 Reader's Guide

If you are familiar with words like fuel map, three–way catalytic converter and the four stroke principle you can omit Section 2. In Section 3 we describe the operating points used for engine testing and how to choose anti aliasing filters and sampling rate. We also define some expressions used later in the thesis. Section 4 is about the simulation models made in Simulink. We model both the engine and the sensors. In Section 5 we discuss conventional air fuel control used today and show simulations and experimental results of implementations. In the section Multi Sensor Lambda Control (Section 6) we start by showing cylinder individual control which utilizes all available information from the sensors. Later on in the same section we modify the control structure to handle an unknown offset in the measured input signals. Both Section 5 and Section 6 show simulations and experimental results. In Section 7 we verify the multi sensor controller by changing to different types of fuel injectors. At the end of each section we give some conclusions. Possible extensions are discussed in Section 8. Conclusions for the entire thesis are presented in Section 9. Appendix A describes the laboratory facility used for engine tests. In Appendix B some larger simulation models are shown and in Appendix C we find the complete controller code.

To make it easier to compare the performance of the presented control structures we show figures at one operating point, namely at 1500 rpm. All operating points data are shown in a table after each section. Simulation results are only shown in tables.

1.5 Previous Related Work

A lot of work has been done in trying to improve lambda control, both sensor construction and regulator algorithms. Some years ago there were no other sensors than the discrete EGO sensor available and work was towards reconstructing lambda from the discrete sensor signal. The UEGO sensors were introduced a couple of years ago but are still relatively expensive, compared to the EGO sensors, for manufacturers of production–line automobiles. Research has also been made on the open loop control (fuel map) using more input parameters. New ideas are often first tested on a single cylinder engine running at low engine speed and with light load. Some researchers use crankshaft angle based sampling instead of time based sampling.

Below some examples of research topics, considering lambda control, are described. Characteristic for all the cited articles and dissertations are the need of a good model to achieve really good results.

In [13] it's described how to use a self tuning regulator (STR) in feedback control from a continuous lambda sensor. They show that the controllability of lambda can be improved by using an STR to compensate for changes in engine dynamics.

An article that describes how to control cylinder by cylinder lambda using two continuous sensors in a V6 engine is [5]. They use a rather complex model based regulator which is able to estimate the cylinder by cylinder lambda from the sensor signals. This requires an algorithm for estimating lambda that, unfortunately, only works at low engine speeds, 800–1200 rpm, due to the sensor time constant. To enable higher engine speed operation some kind of signal processing is needed to compensate for sensor dynamics. The regulator also has feedforward from throttle angle to compensate for transients.

Another article dealing with cylinder individual lambda feedback control is [7]. The main idea in this article is the possibility of using an observer to estimate each cylinder lambda from a measurement slightly downstream the confluence point. To succeed with this, one needs a model of the mixing behavior of exhaust gases from each cylinder at the exhaust manifold confluence point. The estimated cylinder lambda values are then fed to cylinder individual PID controllers. They also use a model to compensate for sensor dynamics. The method was tested on a four cylinder 2.2 liters engine running at 1000 rpm.

For the interested reader there are lots of articles to read. A popular topic is adaptive lambda control, see [3] and [15]. An interesting example of using sliding control is [11]. How to deal with transient errors in A/F are described in [8] and [2].

2 Automotive Fundamentals

2.1 History

Back in the early 1980s almost every automobile engine had a carburetor to meter the air mass flow to fuel mass flow ratio (A/F ratio). It's important to keep the A/F ratio in the vicinity of 15:1 to achieve good engine performance and low emissions. New federal exhaust pollution regulations in California made it hard for the car manufacturers to fulfill these regulations using the old carburetor technique. The change to fuel injectors and electronic fuel control made it possible to use feedback control to meet the new demands.

Due to even more decreasing levels of allowable exhaust pollutants the requirements for accurate A/F ratio control have become very strict over the last decade. A catalytic converter was introduced in the late 1970s to lower exhaust emissions. Especially the three–way catalytic converter made it possible to dramatically reduce the exhaust emissions, see Section 2.4.

The best commercial system of today uses a three–way catalytic converter combined with fuel injectors and an A/F ratio sensor to perform acceptable feedback control.

To approve new automobiles they are tested in different test cycles e.g. USA–FTP 75 and EEC/EU. The test cycles are designed to imitate real driving conditions i.e. city driving and highway driving. During the tests the exhaust gases are collected in sample bags. A direct comparison of the various emissions regulations is made difficult by differences between test cycles and associated differences in engine load. However, it can be maintained that the most stringent of current limits are those applied in USA.

2.2 Main Parts for A/F Control in an SI Engine

Below are the most important physical components, concerning A/F control, and their locations in the engine described together with a brief explanation of the four stroke principle.

Air enters the cylinder via the throttle and the inlet manifold, see Figure 2.1. When the throttle is closed we say that the throttle angle is zero and a wide open throttle means 90° throttle angle. To fully understand an engine cycle we split the process into four stages. First we have the induction stroke.

Induction stroke:

Intake valve: *opened*
Exhaust valve: *closed*
Piston travels: *downwards*
Combustion: *none*

Now air can flow into the cylinder and fuel is injected by a fuel injector. When the piston reaches bottom dead center the inlet valve closes and the process moves into the second stage.

Compression stroke:

Intake valve: *closed*
Exhaust valve: *closed*
Piston travels: *upwards*
Combustion: *initial ignition phase*

About 30–40 crankshaft degrees before top dead center the compressed air fuel mixture is ignited by the spark plug, for more details see [4]. The third stage is the power stroke.

Power stroke:

Intake valve: *closed*

Exhaust valve: *closed*

Piston travels: *downwards*

Combustion: *completed*

The high pressure created by the burning mixture forces the piston downward. It's only during this stroke that actual power is generated by the engine. When the piston, for the second time, reaches bottom dead center the exhaust valve opens. Now we have reached the final fourth stage.

Exhaust stroke:

Intake valve: *closed*

Exhaust valve: *opened*

Piston travels: *upwards*

Combustion: *none*

The upward moving piston forces burned gases from the cylinder through the exhaust manifold into the catalytic converter and out the tailpipe into the atmosphere. Remember that one complete engine cycle requires two complete crankshaft rotations of 360° each.

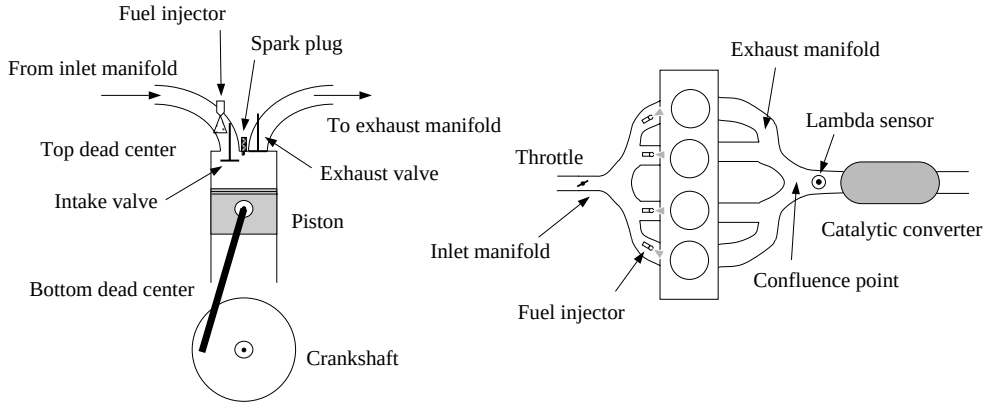

Figure 2.1. Principle sketch of a modern SI engine.

2.3 What is λ ?

The A/F ratio is defined as the ratio between air mass flow, $\dot{m}_a$, and fuel mass flow, $\dot{m}_f$.

$$\text{Air/Fuel ratio (A/F)} = \frac{\dot{m}_a}{\dot{m}_f}$$

Stoichiometry is the optimal mixture of air and fuel in which, when ignited, all of the carbon and hydrogen would completely burn, yielding only carbon dioxide and water. The stoichiometric value depends on the quality of the gasoline, but is normally between 14.57 and 14.70[2]. In other words, about 10000 liters of air are required to support combustion in one liter of fuel!

[2] From now on we will use 14.57 as the stoichiometric value.

The parameter λ (lambda) is the A/F ratio normalized with the stoichiometric value which yields:

$$\lambda = \frac{\frac{\dot{m}_a}{\dot{m}_f}}{(\frac{\dot{m}_a}{\dot{m}_f})_s}$$

When λ exceeds 1 the mixture is called lean and in the same way it's called rich when λ is lower than 1. Under normal driving conditions λ varies from about 0.8 to 1.25. A really lean mixture ($\lambda > 1.3$) ceases to be ignitable and misfire starts to occur. Maximum power is produced under a rich mixture when λ is between 0.8 and 0.9. To reach the economy range in Figure 2.2, without loosing drivability, one has to redesign the engine for lean–burn purposes.

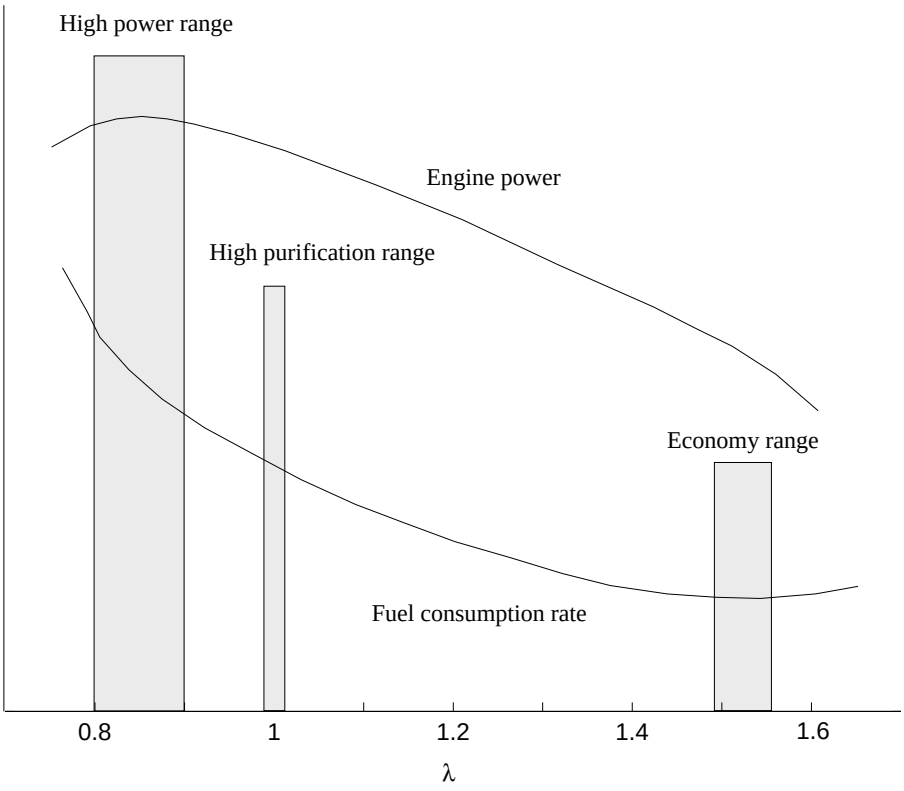

Figure 2.2. Engine power and fuel consumption rate versus λ.

Different Driving Modes

During a ride the engine operates in many different driving modes. Below is a description of the most commonly used modes:

- Cold start occurs when starting a cold engine. Since the engine isn't warm, fuel will condensate on the walls in the cylinders which require more fuel to be injected. Also if the input air temperature is low problems with vaporization of the fuel must be considered.
- Part–throttle operation is the normal driving mode. Now we have a compromise between low fuel consumption and pollutions. Regulations require a low pollution

grade which implies a precise control of λ around the stoichiometric value, see Section 2.4.

- With acceleration we mean suddenly opened throttle valve. The engine should respond by providing its maximum torque. Figure 2.2 shows a necessity to enrich the air–fuel mixture to $\lambda \approx 0.85$.

2.4 Three–Way Catalytic Converter

The main task for a catalytic converter is to reduce the emissions of at least one of the following substances: carbon monoxide (CO), hydrocarbons (HC) and oxides of nitrogen (NO_x). One wants to remove CO because it's a toxic gas and HC and NO_x because they conduce to producing smog.

There is one catalytic converter, the three–way catalytic converter (TWC), that can, when operated under certain conditions, simultaneously remove all three pollutive components to a high degree.

Operating Conditions

Optimal operating temperature for the catalytic converter is between 400–800 °C but catalytic conversion starts at 250 °C. If operating temperatures above 800 °C are used the thermal aging increases severely. In general, a catalytic converter can under ideal operating conditions have a service life of up to 100000 km, see [4] p 25.

The Narrow Operating Window

The above mentioned certain conditions for the TWC are illustrated in Figure 2.3. What one can see is that λ must be held within a very narrow window at the stoichiometric ratio. The width of the window is, for 80 % conversion efficiency for all three pollutants, approximately 0.7 % in λ. This accuracy of the λ control is today practically impossible to achieve without using sophisticated model based controllers in a laboratory environment. Practical experience shows that the narrow window can be broadened to 7 % in λ if oscillations are introduced in λ. The frequency of these oscillations should be about 0.5–1 Hz, see [10] pp 655.

2.5 Electronic Control Unit, Sensors and Actuators

Below we will describe how the electronic control unit performs fuel control using fuel injectors and lambda sensors.

Electronic Control Unit

The electronic control unit (ECU), see Figure 2.4, is a microprocessor based controller with a real time system. Examples of input signals are:

- throttle valve position, α.
- inlet manifold absolute pressure, p_{man}.
- engine coolant temperature.
- engine speed, rpm.

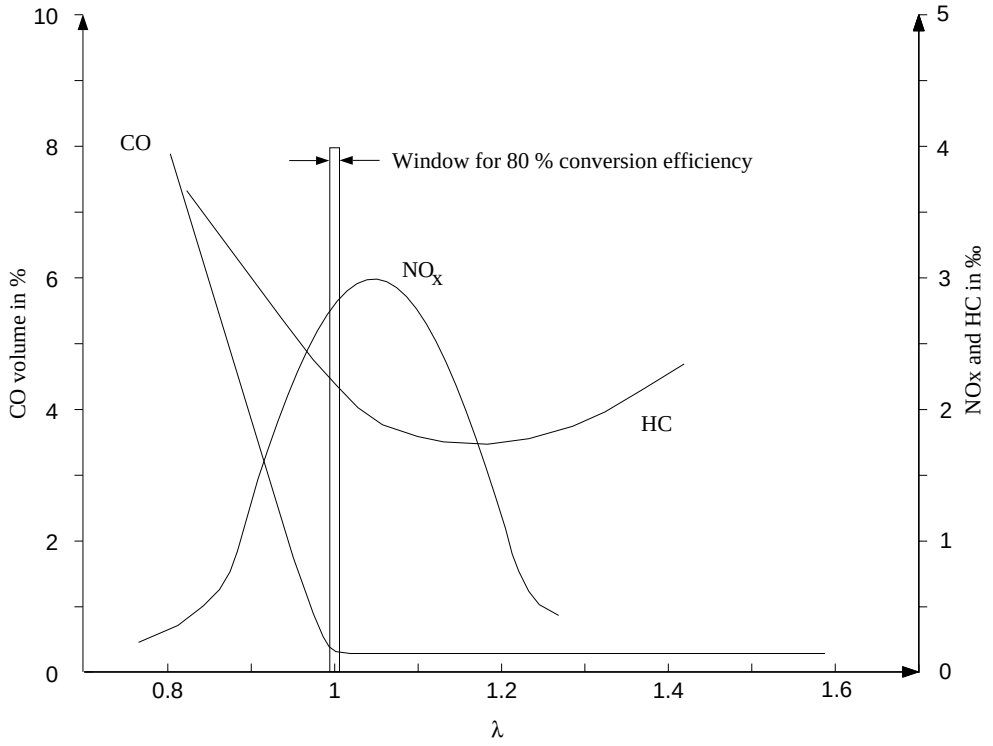

Figure 2.3. The narrow operating window for a TWC. Values of CO, NO_x and HC are shown before a catalytic conversion has occured.

- exhaust gas recirculation, EGR, valve position.
- signals from oxygen sensors.

There are of course also output signals to actuators[3] like:

- fuel metering control.
- ignition control and timing.
- exhaust gas recirculation control.

Figure 2.4. Principle sketch of the ECU with sensors and actuators.

Among many other things the ECU performs calculation of correct amount of fuel to be injected into the cylinders. It starts with a basic injection time found in a look up table (fuel map) using engine speed and inlet manifold absolute pressure as input

[3] A mechanism for moving or controlling something indirectly instead of by hand e.g. fuel injectors and servo motors.

parameters. A basic injection time is always found in the fuel map[4] and since we don't have a feedback signal we call it open loop control.

During some driving modes (e.g. part–throttle operation) the fuel injection system works in a closed loop using the lambda sensor output as a feedback signal. The feedback signal is only allowed to correct the basic injection time by approximately 25 %. The reason to use feedback is to correct for offsets that might occur due to engine wear or different environmental conditions.

Sensors

There are primarily two types of lambda sensors available which differ in price and performance. They both work by measuring the oxygen concentration (they are sometimes called oxygen sensors) in the exhaust gases. The amount of oxygen after combustion is uniquely coupled to the air fuel mixture and we can therefore gain information about lambda by studying exhaust gases.

The exhaust gas oxygen (EGO) sensor is a discrete sensor with switching point around the stoichiometric value, see Figure 2.5. It gives a high voltage signal (≈ 0.9 V) when $\lambda < 1$ and a low voltage signal (≈ 0.1 V) when $\lambda > 1$. Since the EGO signal behaves like a relay we can only sense a lean or rich air fuel mixture. Almost every modern SI engine equipped automobile uses an EGO sensor because it gives acceptable performance at a low cost.

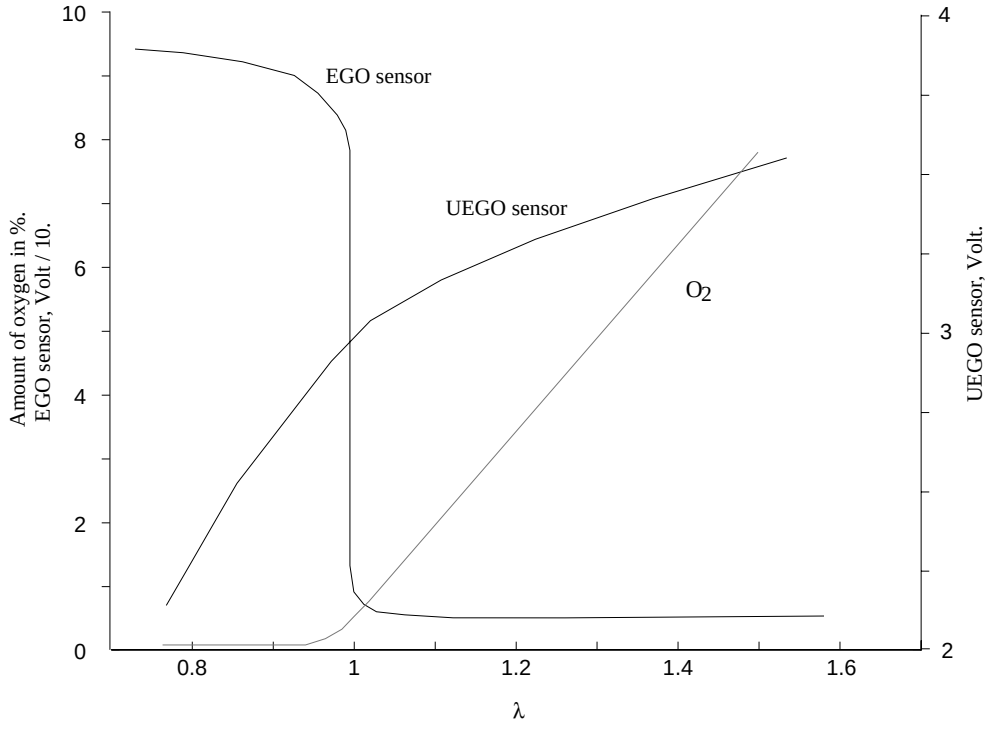

Figure 2.5. Characteristics for EGO and UEGO sensors.

The other type of sensor is the universal exhaust gas oxygen (UEGO) sensor, sometimes called a linear oxygen sensor. We note that the behavior is continuous and

[4] Usually this basic injection time is compensated for low battery voltage and low temperatures.

we can measure an absolute lambda value over the engines whole operating range ($0.8 < \lambda < 1.3$).

Both EGO and UEGO sensors need an operating temperature above 400 °C. Most of todays sensors are preheated to quickly reach an acceptable working temperature, but it's still one of the reasons to use open loop control during cold starts.

Lambda sensors don't instantaneously adjust to a change in air fuel mixture. That because of two reasons: first the sensor has a time constant of about 0.1 s, then one has to consider the transport delay when exhaust gases travel from the cylinder to the location of the sensor. Thus we can model a linear lambda sensor as

$$\frac{1}{1+0.1s}e^{-st_{del}}.$$

We see that simply by moving the sensor further upstream the exhaust manifold there is a possibility to achieve faster lambda control.

Fuel Injectors

Each fuel injector is essentially an electrically actuated valve. When activated the valve opens and fuel is sprayed into the air flowing into the cylinder. The time a fuel injector is open is determined by the duration of the activation signal. Because the fuel pressure is constant, the amount of fuel actually injected by the fuel injector depends solely upon the valves open time (injection duration). Thus the ECU can perform fuel calculation using fuel time rather than fuel volume.

The injection time is in the range $4 < t_{inj} < 10$ ms. Opening of the valve isn't instantaneous, it has a time delay of about half a ms. This delay may vary with the fuel injectors age due to wear.

3 Opening Experiments

Here we present the results from some opening experiments like open loop control and step response tests. We also describe the chosen operating points, some definitions and the Ziegler–Nichols rules of thumb.

3.1 Operating Points

Three operating points are chosen. They are characterized by (speed, p_{man}). The three points are presented in Table 3.1. In this table we also present the corresponding values for engine torque, M, throttle angle, α, and total fuel injection time, t_{inj}, giving $\lambda \approx 1$. The operating points are chosen in such a way that they cover the most interesting parts

speed [rpm]	p_{man} [kPa]	M [Nm]	α°	t_{inj} [μs]
1500	60	80	25	6750
2250	40	50	26	4500
3000	60	100	34	7750

Table 3.1. The chosen operating points.

of the engines operating range. The reason for not choosing a higher speed is that when running at high speed ($\geq$ 3500 rpm) the used electronic throttle control is unreliable. From now on the operating points are called 1500, 2250 and 3000 for convenience. The steps in α during closed loop control are 3° around the values given in Table 3.1.

3.2 Open Loop Control

During open loop control no lambda feedback is used and the injection times are determined only by the fuel map values. The goal with the project is to eliminate differences between cylinders and since differences in our test engine are small (4 % in stationarity and 10 % during transients in lambda) we add offsets to the actual fuel injection times making some cylinders lean and others rich. In Table 3.2 we see the chosen offset times. They result in cylinder individual lambdas between 0.93 (for cylinder 1) and 1.11 (for cylinder 3) in stationarity. At the confluence point lambda tends to be slightly rich. To keep lambda at the same values independent of operating point, different offsets in

Operating point	Cylinder 1	Cylinder 2	Cylinder 3	Cylinder 4
1500	600	250	−500	600
2250	500	250	−250	250
3000	500	250	−500	500

Table 3.2. Offset fuel injection times, in μs.

injection times are used. Figure 3.1 shows lambda values during open loop control with offset times added at 1500 rpm. This is a typical lambda behavior. Notice the differences between single cylinders and that the mean value for lambda at the confluence point is far from 1.

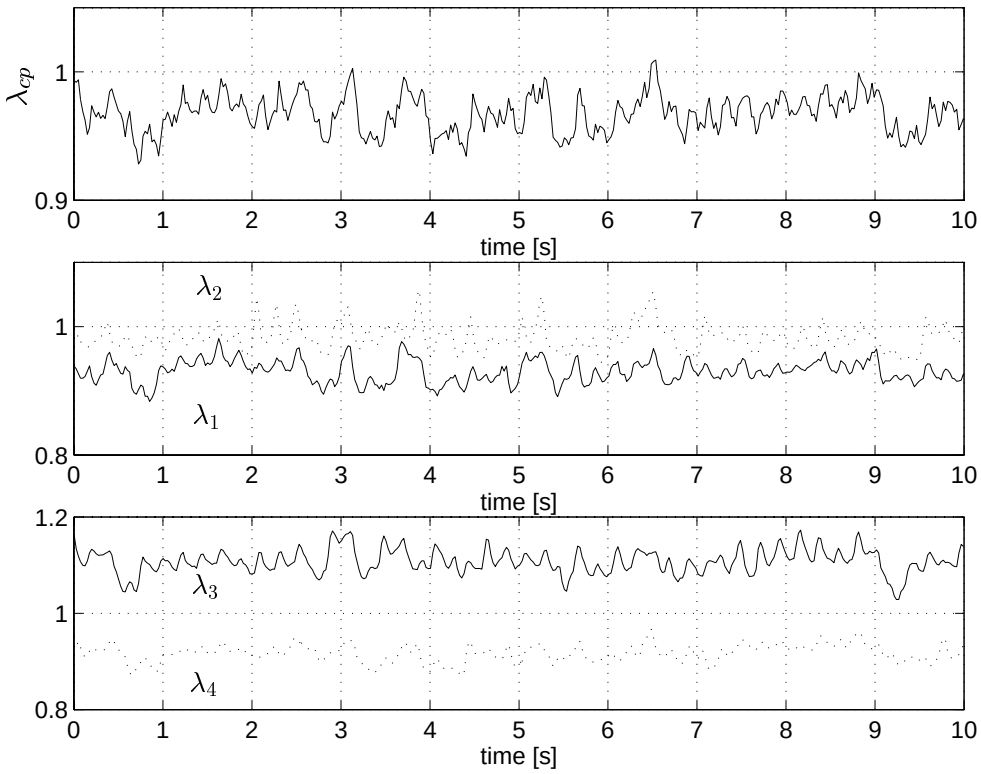

Figure 3.1. Open loop control with offsets in the fuel injection times at 1500 rpm. Top: Lambda at the confluence point. Middle and bottom: Cylinder individual lambda values.

When we in Section 5 and Section 6 evaluate new controllers and control structures we lock the fuel map which means constant basic injection time[5]. This is actually not so realistic since in reality one always uses new fuel map values for each injection. However, when comparing different controllers a constant basic injection time is convenient. Another reason for locking the fuel map is that the picked fuel map values differ very much from cycle to cycle due to noisy inlet manifold pressure measurements.

3.3 Choosing Sampling Rate

To gain information about the system some step response tests are performed. By studying the UEGO sensor step response when making changes in fuel injection time one can determine a cut–off frequency. The steps are between 0.5 and 1 ms which result in λ variations from 0.94 to 1.06. The rise times are in the range of 0.14 s (2250 rpm) to 0.22 s (3000 rpm). A rule of thumb, see [12], is to place between 4 and 8 sampling points on the step response. Testing all three operating points and considering the high noise level at high frequencies we choose 40 Hz as sampling rate corresponding to at least 6 points on the interesting part of the step response. Since the rise times don't differ too much between the operating points we use the same sampling rate at all operating points. Frequency response tests show that sampling at 25 Hz is good enough for the

[5]The ignition angle is also locked.

EGO sensor but since over–sampling isn't a big problem we use the same sampling rate for all sensors and operating points.

3.4 Choosing Filters

Before a signal enters the controller it's filtered through both a digital and an analog filter. They are used for anti aliasing and noise purposes. The analog filter is a low pass RC–lag. Experience showed that the cut–off frequency isn't of great importance as long as it's higher than the sensors bandwidth! Of course the cut–off frequency must be lower than half the sampling rate. Since the lambda signal doesn't contain information above 10 Hz, see Heywood [10], and since we use standard resistor and capacitor components the cut–off frequency becomes 159 Hz. Because noise enters the system both before and after the analog filter we use a high sample rate, 2000 Hz[6], and filter digitally with a 2:nd order Butterworth filter with a cut–off frequency of 20 Hz, half the controllers sampling rate. Using a 2:nd order filter is suitable, a higher order would lead to a too high phase lag. To summarize we use a low pass RC filter ($f_c = 159$ Hz), sample with 2000 Hz and filter digitally with a Butterworth filter ($f_c = 20$ Hz). The output signal from the Butterworth filter is decimated at the controller sampling rate, 40 Hz. These choices of filters give a noise level of about 7 mV corresponding to 0.6 % in λ under lean conditions and 0.3 % under rich conditions for the UEGO signals. The difference between rich and lean operation depends on the nonlinearity in the UEGO sensor characteristic. The EGO sensor signal isn't affected much due to noise since the difference in voltage between rich (0.9 V) and lean (0.1 V) is much bigger than the noise amplitude.

3.5 Ziegler–Nichols Rules of Thumb

To find PI parameter values for the control structures involving UEGO sensors the Ziegler–Nichols rules of thumb, see [6], are applied. The controller (PI) is assumed to be on the following form:

$$e(t) = r(t) - y(t)$$

$$u(t) = Ke(t) + \frac{K}{T_i}\int_0^t e(\tau)d\tau$$

Let $T_i = +\infty$. Increase K until a stable self oscillation occurs. Note the corresponding K–value, K_0, and the period, T_0. The proper controller parameters are then determined as

$$K = 0.45K_0$$

$$T_i = T_0/1.2.$$

A disadvantage with the above controller parameterization is that it's unable to contain only an integral part. By re–parameterization

$$u(t) = Ke(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau \tag{3.1}$$

[6] The controller can't sample faster without overload.

we get rid of that problem. The rules of thumb are then:

$$K = 0.45K_0$$

$$T_i = \frac{T_0/1.2}{0.45K_0} = \frac{T_0}{0.54K_0}$$

When the Ziegler–Nichols rules of thumb are applied, no offsets are introduced in order to simulate a new engine. Furthermore the fuel map is locked.

3.6 Some Definitions

Some of the definitions used in the evaluations in the following sections are described below.

By sc, single cylinder, we mean the maximum or minimum value found in any cylinder. By cp we mean the exhaust manifold confluence point.

The time measurement T_1 is the time it takes to force λ back into the interval $[0.96, 1.04]$ after a positive step in throttle angle has been applied. In a similar way T_2 is related to a negative step. Note that if an overshoot occurs this results in a longer T_1 or T_2 i.e. T_1 and T_2 are the times to reach a λ in the above mentioned interval in stationarity.

The rms value is defined as

$$\text{rms} = \frac{1}{\sqrt{n}}\sqrt{\sum_{i=1}^{n}(\lambda_{cp}(i) - 1)^2} \cdot 100\%$$

4 Finding a Simulation Model

We need some kind of simulation model to test our control principles on and to find out how to choose controller parameter values before we carry on with experimental tests on a real engine. To find a simulation model we first derive the necessary equations to calculate λ from fuel injection time and air mass flow, see Section 4.1 and Section 4.2. The second step includes finding proper model parameter values and forming Simulink blocks for a single cylinder, see Section 4.3, and for different sensors, see Section 4.4. For more information about Simulink see [1].

4.1 Mean Value Models of SI Engines

If one wants to model a four stroke SI engine completely, it requires a model that handles e.g. non linearities in sensors and engine dynamics, fuel injections taking place at discrete moments and cycle to cycle[7] variations. A class of models which is intermediate between large cyclic simulation models and empirical transfer function models is mean value models, described in [9].

As the name indicates the model is restricted to describe the mean values of the engine variables. Time scale is much longer than required for a single engine cycle but small enough to describe the most rapidly changing engine variables. Relationships between engine variables where it takes only a few engine cycles to reach equilibrium are expressed with static equations. On the other hand, relationships where it takes between 10 and 1000 cycles to reach equilibrium are expressed with differential equations.

Three states are needed in the mean value model of the SI engine namely: fuel film mass flow, $\dot{m}_{ff}$, crankshaft speed, n (in Hz), and inlet manifold absolute pressure, p_{man}. The engine input variables are throttle angle, α, injected fuel mass flow, $\dot{m}_{fi}$, and ignition timing angle. Engine load, power or torque, is treated as disturbances of the engine. Since we are only interested in fuel dynamics we only derive the proper equations.

The equations that are needed to describe the behavior in λ are derived below. Equation 4.1 tells us how to calculate the injected fuel mass flow, $\dot{m}_{fi}$, from the calculated total fuel injection time, t_{inj}. The parameter t_0 is fuel injector dead time, n is crankshaft speed and k is a conversion factor from fuel injection time to fuel mass flow. The factor 1/2 originates from that a cycle corresponds to two crankshaft rotations.

$$\dot{m}_{fi} = \frac{n}{2} k (t_{inj} - t_0) \tag{4.1}$$

The injected fuel mass flow splits in two parts, vaporized fuel and fuel film on the manifold walls. Vaporization is so fast that a static equation will do, thus:

$$\dot{m}_{fv} = (1 - X)\dot{m}_{fi} \tag{4.2}$$

The remaining part, X, becomes fuel film in the manifold:

$$\dot{\bar{m}}_{ff} = \frac{1}{\tau_f}(-\dot{m}_{ff} + X\dot{m}_{fi}) \tag{4.3}$$

Adding fuel vapor mass flow, $\dot{m}_{fv}$, and fuel film mass flow, $\dot{m}_{ff}$, gives the cylinder port fuel mass flow, $\dot{m}_f$, that is:

$$\dot{m}_f = \dot{m}_{fv} + \dot{m}_{ff} \tag{4.4}$$

[7] With a cycle we mean the four strokes i.e. two crankshaft rotations.

Combining Equation 4.2, 4.3 and 4.4 yields:

$$\dot{m}_f = \frac{1 + (1 - X)s\tau_f}{1 + s\tau_f}\dot{m}_{fi} \tag{4.5}$$

This equation is also derived in [9]. For a given throttle angle, α, the air mass flow into the cylinder, $\dot{m}_a$, is assumed to be constant. Now we use the definition of λ to calculate λ_c in the gases leaving the cylinder:

$$\lambda_c = \frac{\frac{\dot{m}_a}{\dot{m}_f}}{(\frac{\dot{m}_a}{\dot{m}_f})_s} \tag{4.6}$$

Thus the equations needed to calculate λ_c from t_{inj}, n and $\dot{m}_a$ are 4.1, 4.5 and 4.6.

4.2 Sensor Models

Since the sensor isn't placed directly after the exhaust valve there is a mandatory transport delay, t_{del}. Furthermore, the sensor (EGO and UEGO type) is assumed to have dynamics which can be described by a first order system. Thus:

$$\lambda_m = \frac{\frac{1}{\tau_l}e^{-st_{del}}}{s + \frac{1}{\tau_l}}\lambda_c \tag{4.7}$$

Due to noise and to avoid aliasing we also need a second order low pass filter which finally gives, for UEGO sensors:

$$\lambda_f = \frac{B(s)}{A(s)}\lambda_m \tag{4.8}$$

For the EGO sensor we also need a relay like part:

$$r_f = \frac{B(s)}{A(s)}\text{sign}(1 - \lambda_m) \tag{4.9}$$

The equations needed to calculate λ_f from λ_c are then 4.7 and 4.8 for a UEGO sensor. To calculate the output from an EGO sensor the required equations are 4.7 and 4.9.

4.3 Forming a Simulink Block for a Single Cylinder

A simulation model in Simulink for a single cylinder is given in Figure 4.1. This block is called *cylinder* from now on in the simulation models. Note that in this model we have added some further signals. *Lambda correction time* is the output from the lambda controller, *throttle* is used to simulate changes made by the driver and some noise is introduced to imitate actual driving conditions. Noise in $\dot{m}_a$ and in t_{inj} have frequency $n/2$ corresponding to the injection frequency for a single cylinder. The maximal noise amplitudes are 5 % in $dma0$ and 0.5 ms in t_{inj}. The variation in n is a sine wave with an amplitude of 25 rpm and frequency 0.15 Hz.

Typical parameter values, at 1500 rpm ($n = 25$ Hz), are $X = 0.2$, $tauf = 0.3$ s, $t0 = 0.5$ ms, $tmap = 7.0$ ms, $dma0 = 0.7 \cdot 25/1000 = 0.0175$ and $k = 2 \cdot 0.7/(1000 \cdot 14.57 \cdot (0.007 - 0.0005)) = 1.4/94.705 \approx 0.014$. These values and $throttle = 1$ correspond to $\lambda \approx 1$ under stationary conditions.

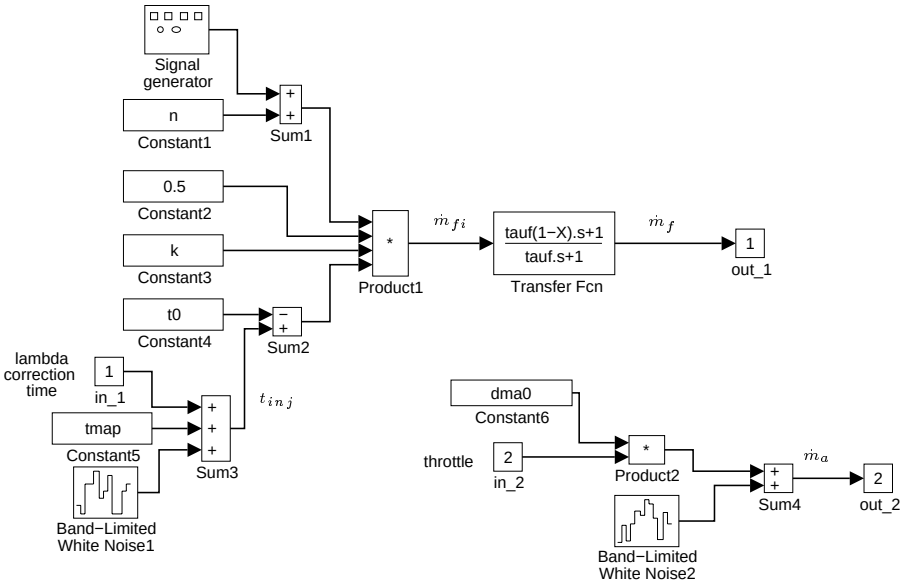

Figure 4.1. Simulation model for a single cylinder as a Simulink block.

4.4 Forming a Simulink Block for a Sensor

A simulation model in Simulink for a UEGO sensor and an anti aliasing filter is given in Figure 4.2.

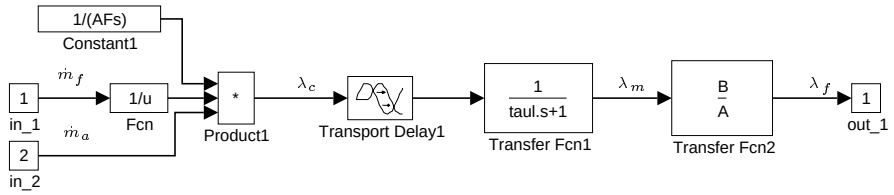

Figure 4.2. Simulation model for a UEGO sensor and an anti aliasing filter as a Simulink block.

By introducing a relay like part, here the signum function, we get the block for an EGO sensor, see Figure 4.3.

A typical time constant for the sensors is $taul = 0.085$ s. For convenience we use the same parameter value for EGO and UEGO sensors. Running at 1500 rpm gives a transport delay in the order $tdel = 0.17$ s (to the confluence point). Furthermore, we assume that the transport delay to the close to cylinder located UEGO sensors, $tdelcyl$, is $\frac{tdel}{5}$. Choosing the filter as a second order low pass Butterworth filter with cut–off frequency 20 Hz gives $B = 15791$ and $A = s^2 + 178s + 15791$. Since the conversion characteristic from volt to lambda for the UEGO sensor is quite well known it isn't a too coarse approximation to assume the UEGO sensor being linear. From now on, these blocks are called *UEGO and filter* and *EGO and filter* in the simulation models.

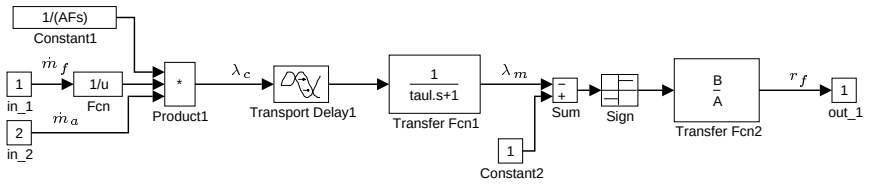

Figure 4.3. Simulation model for an EGO sensor and an anti aliasing filter as a Simulink block.

4.5 About the Simulations

To build a complete simulation model of an engine in Simulink the above mentioned blocks are combined with time discrete PI controllers running with a sampling rate equal to 40 Hz. The controllers are allowed to correct the injection time with a maximum of ± 2 ms.

How to calculate the lambda value measured downstream the confluence point from the cylinder individual measurements is a quite complicated problem. The first approach, assuming that lambda at the confluence point is the mean value of the cylinder individual measured lambdas, isn't good enough because the fuel injection times differ from cylinder to cylinder due to disturbances and offsets. The second approach, assuming that the engine has a transfer function equal to 1, will do, thus:

$$\lambda_{cp} = \frac{cyl\#1\dot{m}_a + cyl\#2\dot{m}_a + cyl\#3\dot{m}_a + cyl\#4\dot{m}_a}{cyl\#1\dot{m}_f + cyl\#2\dot{m}_f + cyl\#3\dot{m}_f + cyl\#4\dot{m}_f} \frac{1}{(A/F)_s}$$

Note that a transport delay from the cylinders to the confluence point is also required.

To find PI parameter values for the control structures involving UEGO sensors the Ziegler–Nichols rules of thumb, see Section 3.5, are applied.

When the Ziegler–Nichols rules of thumb are applied, all cylinders have the same $\dot{m}_a$ and all fuel injectors have the same t_0 in order to simulate a new engine. To imitate differences between fuel injectors and between intake valves, cylinder individual t_0 and $\dot{m}_a$ are introduced representing aging. The cylinder individual values, see Table 4.1, are used during the actual simulations. The values are chosen to give the same effect as the offset times in Section 3.2.

cylinder #	$dma0\cdot?$	$t0\cdot?$
1	$dma0 \cdot 0.9$	$t0 \cdot 1.5$
2	$dma0 \cdot 0.95$	$t0 \cdot 2$
3	$dma0 \cdot 1.1$	$t0 \cdot 2$
4	$dma0 \cdot 0.9$	$t0 \cdot 1.5$

Table 4.1. Cylinder individual parameters used during simulations.

To simulate steps in throttle angle the *throttle*, see e.g. Figure 4.1, is switched between 0.9 and 1.1.

5 Single Sensor Lambda Control

In this section we implement the conventional lambda control used in production automobiles to compare with our designed controllers. Differences between control using EGO and UEGO sensor are shown both in simulations and by experimental results.

5.1 Conventional Lambda Control

By conventional lambda control we mean the combination of a single EGO sensor and an I or PI controller. The reason for studying conventional lambda control is that we need a controller to compare our controllers properties with. First we use only an I controller, limit cycle control, and then we also introduce a P part, jump back control.

Limit Cycle Control

When combining a relay like feedback signal with a system with a pure transport delay and an I controller so called limit cycles occur. Below are the period and amplitude for the oscillations in the controller output derived. To simplify the calculations the engine is assumed to have a transfer function equal to 1 and the EGO sensor is assumed to behave like a perfect relay. The EGO sensor output is 1 when $\lambda < 1$ and 0 when $\lambda > 1$. The transport delay, t_{del}, is in this example 1 s and T_i in Equation 3.1 is 10. The system is illustrated in Figure 5.1. The controller output is a correction to the fuel injection time picked from the fuel map. The air mass flow is constant in this example. The first approach, using describing function, see [6], to calculate the period, doesn't

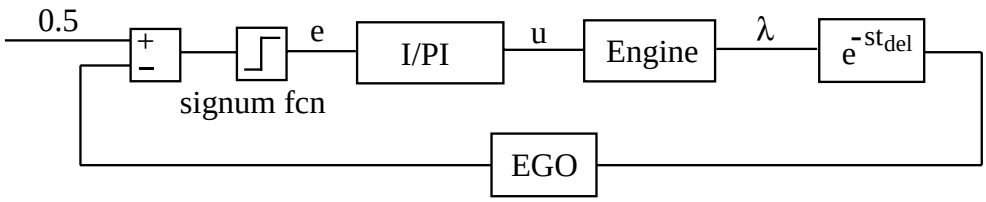

Figure 5.1. A principle sketch of EGO sensor control.

work because the system doesn't have low pass properties. Instead we use a heuristic method to calculate the period and amplitude. In Figure 5.2 λ, the controller output, u, and the EGO sensor output are shown. At A λ switches from rich to lean. When the influence of the lean mixture reaches the EGO sensor at B, it switches to 0 (lean) and the controller starts to enrich the mixture. At C λ switches from lean to rich and again it takes 1 s, corresponding to the transport delay, before the EGO sensor switches at D. The controller now starts to make the mixture more lean and at E λ again switches from rich to lean. This is observed at the EGO sensor at F. In Figure 5.2 we see that the period, T, is equal to four times the time delay

$$T = 4t_{del}. \tag{5.1}$$

This means that the shorter the transport delay is, the higher the frequency of the limit cycles will be. The transport delay decreases as engine speed increases, which implies higher frequencies at higher engine speeds. If the engine and sensor dynamics are faster than the transport delay, Equation 5.1 gives a good approximation of the transport delay.

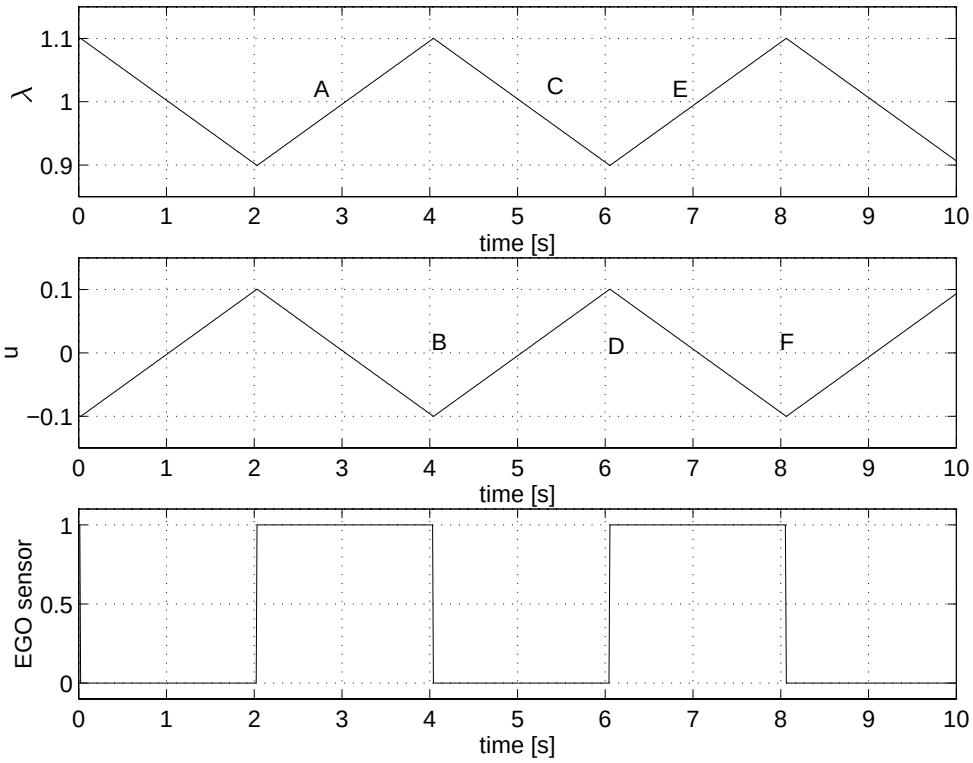

Figure 5.2. Illustration of the limit cycle phenomena.

To calculate the amplitude in the controller output we note that the time the EGO sensor output is constant corresponds to two times the transport delay. During this time the controller output goes from its maximum to its minimum or vice versa, thus

$$u_{p-p} = \frac{2t_{del}}{T_i}e. \tag{5.2}$$

Now we see that by choosing a bigger value of T_i we get a smaller amplitude in u and therefore also in λ. Unfortunately this only holds in theory and not practically due to disturbances and dynamics in sensors and in the engine. The choice of T_i is also a compromise between good transient behavior and small variations in λ under stationary conditions. Since we are mostly interested in the latter we choose a T_i such that choosing a bigger T_i will not give significantly smaller variations in λ at stationarity.

Jump Back Control

To achieve better transient properties we also add a proportional part to the controller. By choosing the proportional part in a smart way we can double the oscillation frequency but still have the same oscillation amplitude. Consider Figure 5.1 and Figure 5.2 again.

The idea is to select K such that u is stepped back to 0 every time the EGO sensor switches. This is illustrated in Figure 5.3. The switching behavior due to the transport delay described for limit cycles also holds here. From Figure 5.3 one can see that the proportional part, $2Ke$, must be equal to the limit cycle amplitude in controller output, $\frac{u_{p-p}}{2}$, since the controller output is $\frac{u_{p-p}}{2}$ just before A and 0 just after A. Note that Equation 5.2 also holds now. The factor 2 in the proportional part is introduced because

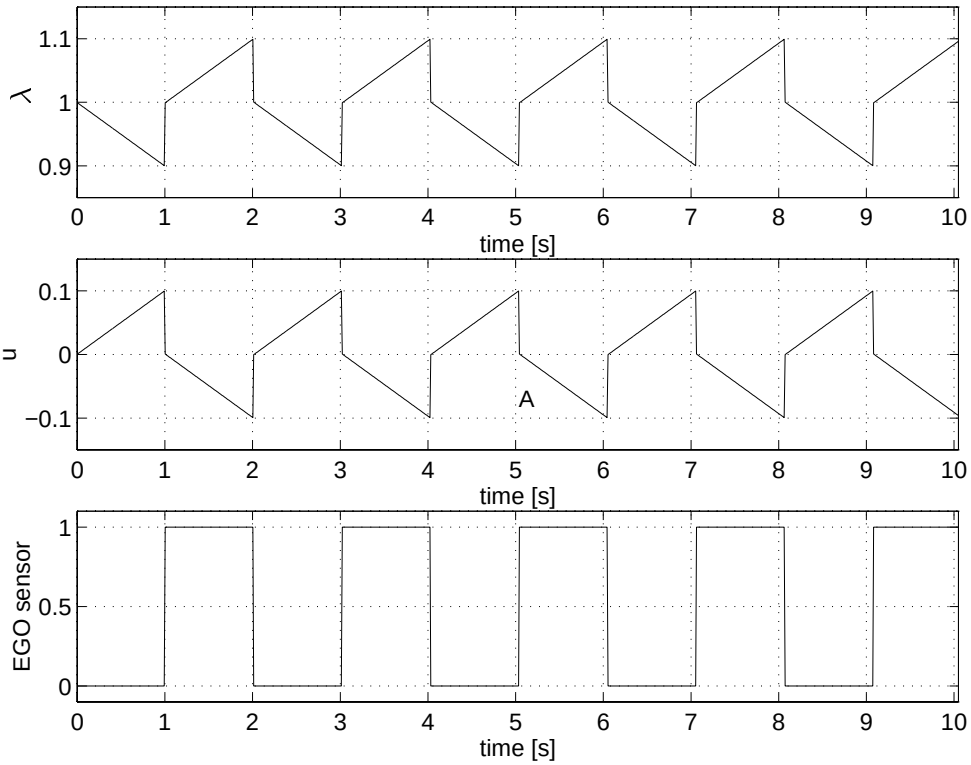

Figure 5.3. Illustration of jump back control.

the signum function output changes with ± 2 when the EGO sensor switches. Thus, to achieve jump back behavior we choose K as

$$K = \frac{t_{del}}{2T_i}. \tag{5.3}$$

As we can see in Figure 5.3 this choice doubles the frequency without any increase in amplitude.

Simulations

To find suitable starting values for the controller parameters to be used in the experiments on the real engine we first make some simulations. The complete simulation model is shown in Appendix B, Figure B.1. Note that the UEGO sensors in this model are used only for validation purposes. The simulation results are summarized in Table 5.1.

It was found that $T_i = 1000$ was suitable. This value gives an average amplitude in λ of about 4 % around the stoichiometric value $\lambda = 1$. The oscillation period is 1.1 s. Using Equation 5.1 gives 0.275 s as an estimate of the transport delay. Remembering that the delay during the simulations was 0.17 s we see that the engine and sensor dynamics are not so fast that they can be neglected. However, when choosing the K value for jump back control we use the estimated value $t_{del} = 0.275$ s in Equation 5.3. This results in a too high K value giving a slight increase in the amplitude under stationary conditions compared to limit cycle control, see Table 5.1.

The period for jump back control is 0.64 s i.e. it isn't halved compared to 1.1 s. From Table 5.1 we also see that rms during transients is reduced but not as much as one might

Operating point	Reg	λ_{max} cp	λ_{min} cp	λ_{max} sc	λ_{min} sc	T_1	T_2	rms %
1500 stationarity	I	1.04	0.96	1.24	0.86	—	—	2.30
1500 step	I	1.25	0.80	1.49	0.73	1.5	1.6	8.51
1500 stationarity	PI	1.06	0.94	1.24	0.86	—	—	2.28
1500 step	PI	1.17	0.82	1.40	0.74	1.1	1.2	6.33

Table 5.1. Summary of simulation results when using conventional lambda controllers.

have expected. These differences between theory and simulation depend mainly on the fact that sensor and engine dynamics are not negligible in this particular case.

Implementation

To implement the control law, rewritten here for convenience, we use the method with adjustment of the integral part, see [14].

$$u(t) = Ke(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau$$

By adjusting the integral part we get rid of the problem with anti wind up. The mathematical expression for adjustment of the integral part is given below.

$$\begin{aligned} I_n &= I_{n-1} + \frac{T_s}{T_i}e_n \\ v_n &= Ke_n + I_n \\ u_n &= \begin{cases} u_{max} & \text{if } v_n > u_{max} \\ v_n & \text{if } u_{min} \le v_n \le u_{max} \\ u_{min} & \text{if } v_n < u_{min} \end{cases} \\ I_n &= I_n + \frac{T_s}{T_t}(u_n - v_n) \end{aligned}$$

In this thesis T_t is chosen as $T_t = T_i$ i.e. tracking.

Now we give pseudo–code for the I and PI controller using an EGO sensor. By checking if the filtered input signal is below or above a reference value (here 0.5 V corresponding to the switching point of the EGO sensor, see Figure 2.5) we make a relay interpretation. Since we sample with 2000 Hz and control with 40 Hz we don't have to enter the inner loop more than every 50:th time. This is done by checking the "control time" flag. Note that T_s corresponds to the controllers sampling rate, 40 Hz.

ReadInput(y);
y_f = Butterworth(y);
if (control time) **then**
    $e = 0$;
    if $(ref - y_f) > 0$ **then**
        $e = 1$;
    else if $(ref - y_f) < 0$ **then**
        $e = -1$;
    $I = I + \frac{T_s}{T_i}e$;
    $v = Ke + I$;

if $(v > u_{max})$ **then**
$u = u_{max}$;
else if $(v < u_{min})$ **then**
$u = u_{min}$;
else
$u = v$;
Out(u);
$I = I + \frac{T_s}{T_i}(u - v)$;
Update filter states;
Wait;

The calculated output signal, u, is added to the basic injection time found in the fuel map. This is done for all the following controller outputs. The complete code is listed in Appendix C, *void EGOcontroller*.

Experimental Results

Now we describe the engine testings of I and PI (jump back) control using an EGO sensor. To fully evaluate the controllers we study both stationary behavior and transient response. Each figure is plotted for the first operating point, 1500 rpm 60 kPa. In Table 5.3 we summarize the results of both I and PI control using an EGO sensor. During all tests the fuel map is locked. To determine T_i we start with the value given by simulations (here 1000) and adjust it until the lambda peak to peak value is at minimum. At 1500 rpm this happens when $T_i = 2000$. In Figure 5.4 and Figure 5.5 we see the result of I control when using an EGO sensor. Notice that the mean value of λ_{cp} now is 1 with fluctuations of about $\pm$ 6 %. In the behavior of λ_{cp} we see the limit cycles and they are even more noticeable in the regulator output signal, u. The middle plot in Figure 5.4 shows how the EGO sensor appears like a relay. Cylinder individual lambdas are pictured in Figure 5.5. They range between 0.9 and 1.2 and differ in level. Remember the impossibility to eliminate differences between cylinders. This holds for all presented single sensor controllers in this section.

We study transient response by introducing throttle angle changes, representing sudden acceleration or retardation. The steps result in changes in engine speed of approximately $\pm$250 rpm. In Figure 5.6 the actual engine response is shown together with the output signal from the I regulator. Here T_1 is around 2.5 s and $T_2 = 3.1$ s. These values depend on engine speed and higher speeds result in most cases in shorter settling times.

If we now consider the PI controller and choose K as described in Equation 5.3 we can take advantage of the jump back phenomena. First we need an estimate of the transport delay, t_{del}, the delay between actual injection and sensor sensing. Using Equation 5.1 for the operating points give the following estimates: 0.28 s at 1500 rpm, 0.27 s at 2250 rpm and 0.17 s at 3000 rpm. Recall that the simulations showed that Equation 5.1 tended to give a too high value for t_{del} resulting in a too high K value. Due to this fact we instead use the following estimates for the transport delay in Equation 5.3, 0.2 s at 1500 rpm and 2250 rpm and 0.15 s at 3000 rpm. A lower value for a higher speed is due to faster exhaust gas transport and faster engine cycles. For a complete list of used regulator parameters, see Table 5.2. The results of engine testing, in stationarity, are shown in Figure 5.7. One notices the higher frequency of the oscillations as expected. The frequency isn't doubled and is a result from the fact that the engine and sensor don't have transfer functions

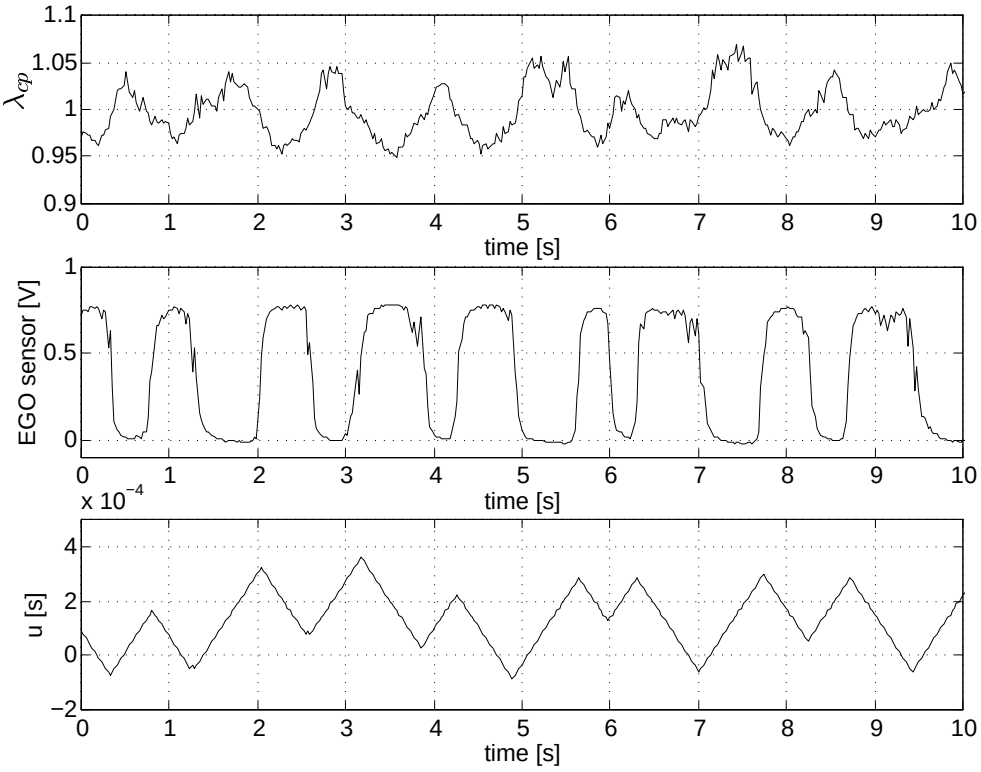

Figure 5.4. EGO sensor and I control in stationarity. Top: Lambda at the confluence point. Middle: Relay behavior of the EGO sensor. Bottom: Controller output signal with characteristic limit cycle behavior.

Operating point	Reg	K	T_i
1500	I	—	2000
2250	I	—	2000
3000	I	—	750
1500	PI	1/20000	2000
2250	PI	1/20000	2000
3000	PI	1/10000	750

Table 5.2. Controller parameters used during conventional lambda control experiments.

equal to 1. When viewing the controller output signal we notice the characteristic jump back behavior. The fluctuations in lambda are still around ±5 % and lambda has still a mean value equal to 1. Making throttle changes resulted in Figure 5.8. Somewhat lower amplitudes are achieved compared to the previous I controller.

We are now ready for a short summary of the results concerning conventional lambda control. By studying Table 5.3 we see large deviations in λ_{max} and λ_{min} from the reference value for single cylinders. This is promising for the multi sensor implementation where we should be able to control single cylinders. Lambda values at the confluence point don't differ much between I and PI control. A slight but significant improvement is achieved by using PI control concerning the rms value. Still, the most noticeable achievements are radically improved settling times at low engine speeds.

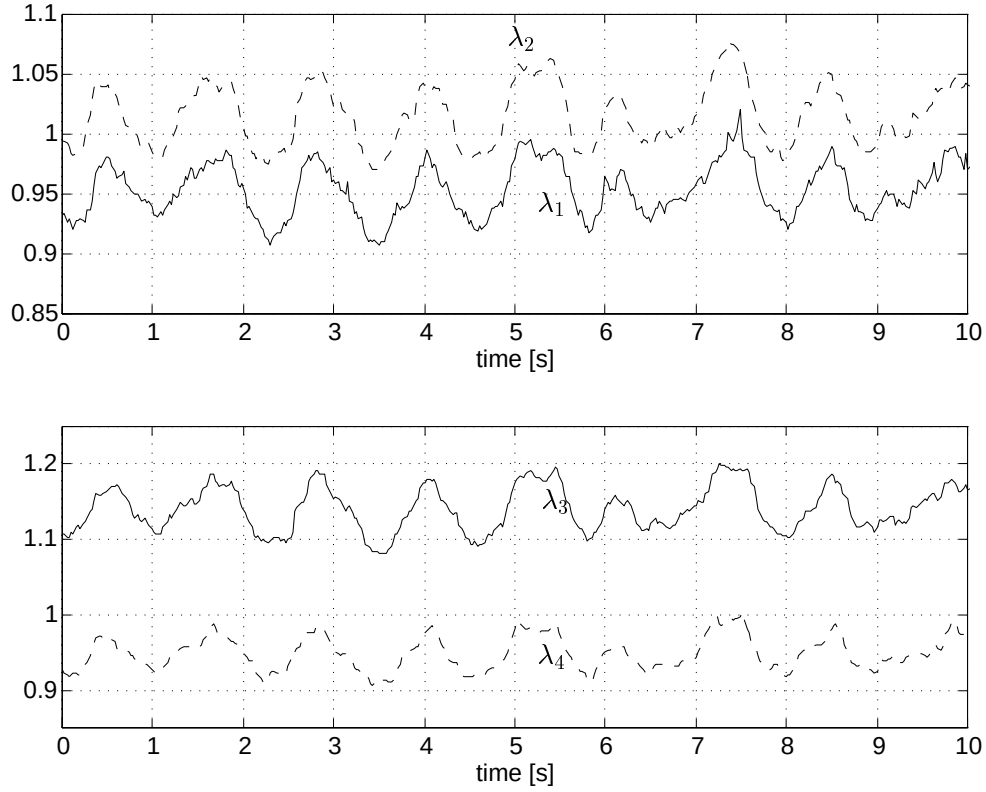

Figure 5.5. Cylinder individual lambda values in stationarity when controlling with an I controller and an EGO sensor.

Operating point	Reg	λ_{max} cp	λ_{min} cp	λ_{max} sc	λ_{min} sc	T_1	T_2	rms %
1500 stationarity	I	1.07	0.95	1.20	0.91	—	—	2.69
2250 stationarity	I	1.04	0.96	1.17	0.90	—	—	1.69
3000 stationarity	I	1.04	0.96	1.14	0.92	—	—	1.84
1500 step	I	1.30	0.84	1.52	0.78	2.5	3.1	11.20
2250 step	I	1.38	0.87	1.77	0.82	2.3	1.8	11.50
3000 step	I	1.21	0.88	1.39	0.82	1.0	0.8	5.60
1500 stationarity	PI	1.07	0.95	1.21	0.91	—	—	2.56
2250 stationarity	PI	1.03	0.95	1.17	0.91	—	—	1.66
3000 stationarity	PI	1.03	0.96	1.13	0.92	—	—	1.55
1500 step	PI	1.27	0.86	1.48	0.81	1.6	1.6	8.66
2250 step	PI	1.25	0.84	1.68	0.78	1.8	2.1	8.45
3000 step	PI	1.18	0.86	1.32	0.81	0.8	1.0	5.07

Table 5.3. Summary of experimental results when using conventional lambda controllers.

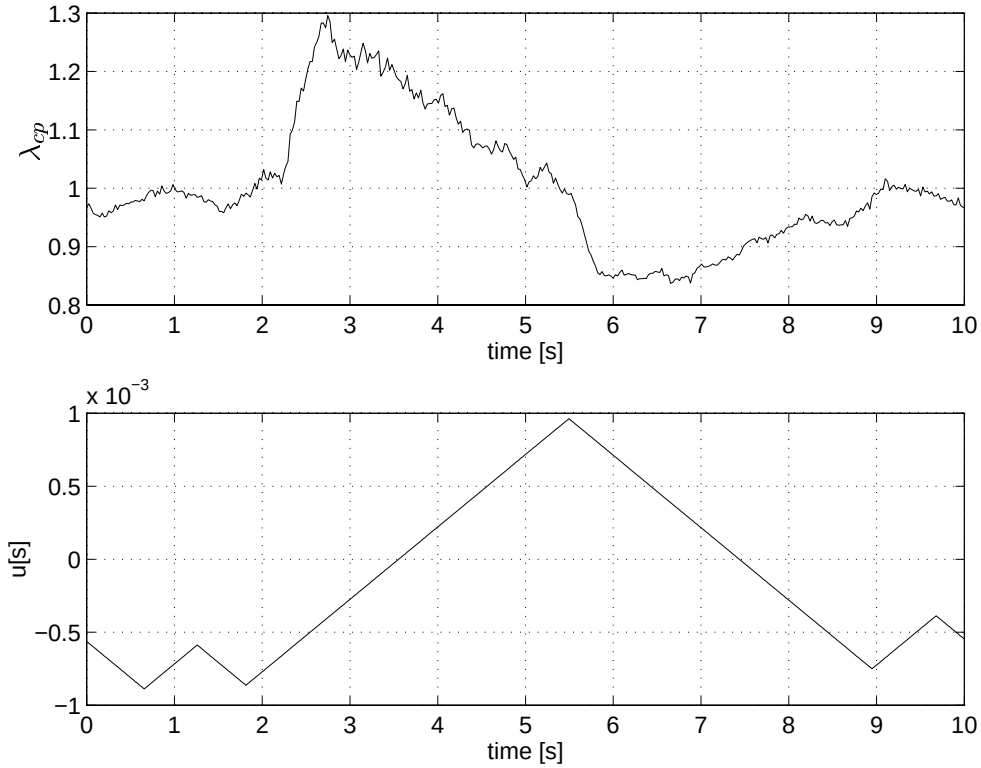

Figure 5.6. EGO sensor and I control during transients. Top: Lambda at the confluence point. Bottom: Controller output signal.

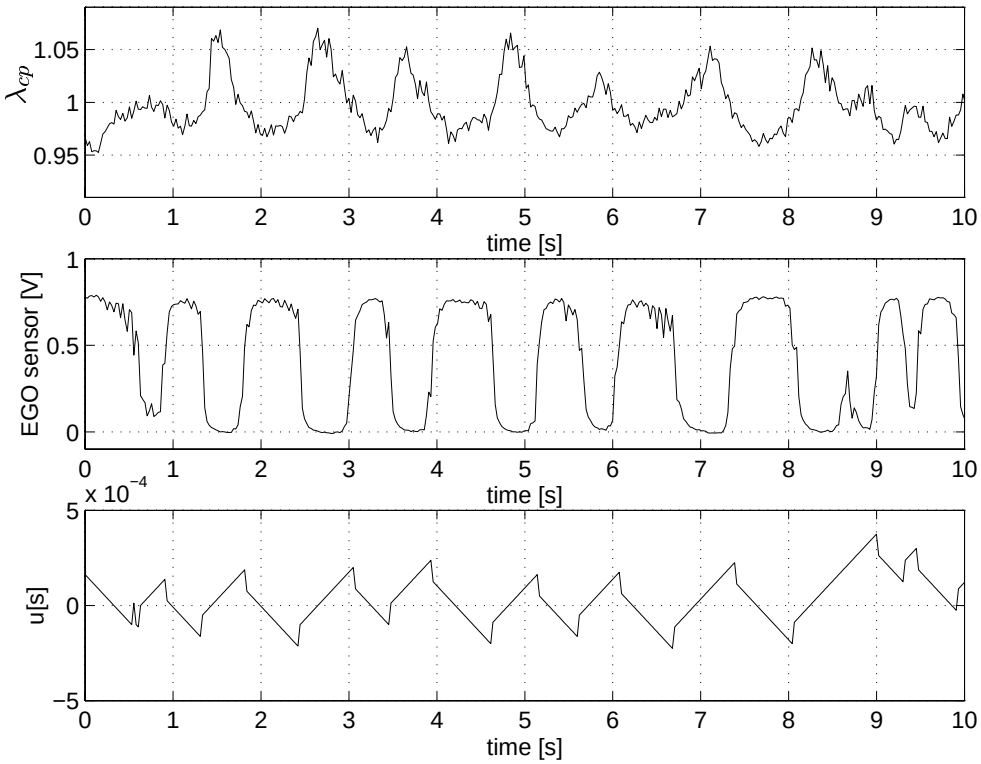

Figure 5.7. EGO sensor and PI control in stationarity. Top: Lambda at the confluence point. Middle: EGO sensor signal. Bottom: Jump back phenomena in controller output signal.

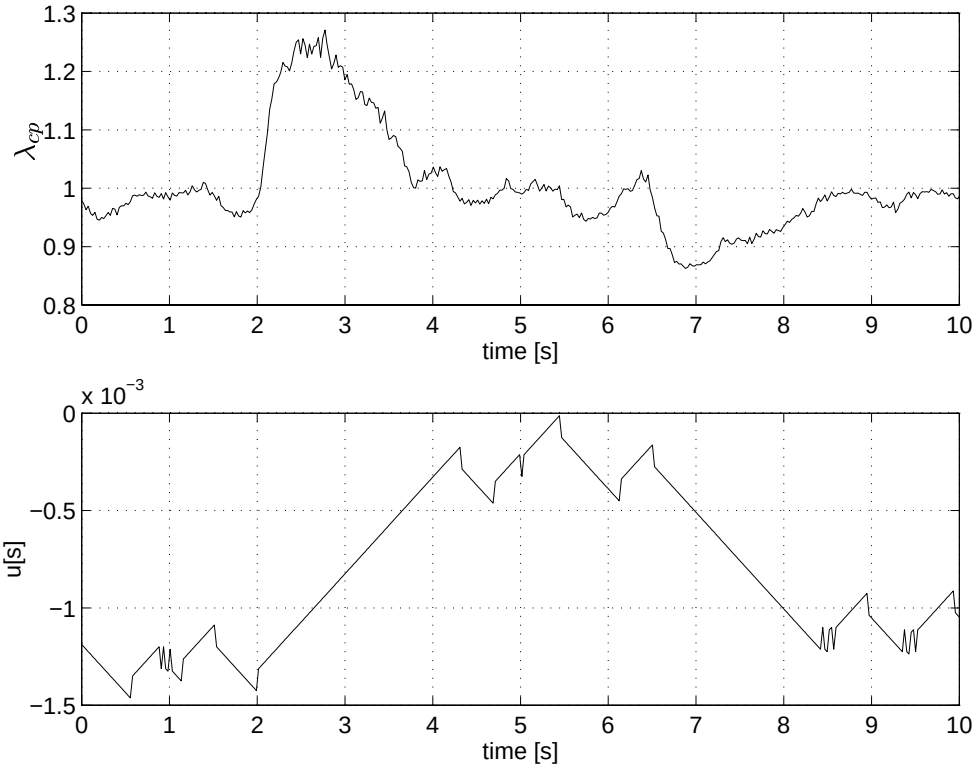

Figure 5.8. EGO sensor and PI control during transients. Top: Lambda at the confluence point. Bottom: Controller output signal.

5.2 Lambda Control Using One UEGO Sensor

When using a UEGO sensor we are able to get absolute lambda values over the entire engine operating range. More information about the process should lead to better controlling results. Phenomenas like limit cycles and jump back are no longer present. By not relying on a switching point we are able to give the regulator an input reference in lambda not equal to 1, although $\lambda_{ref} = 1$ is used during both simulations and engine testing.

The Control Structure

Figure 5.9 shows the control structure used in this section for lambda control with a UEGO sensor. We use $1/\lambda$ rather than λ as an input signal because $1/\lambda$ is proportional

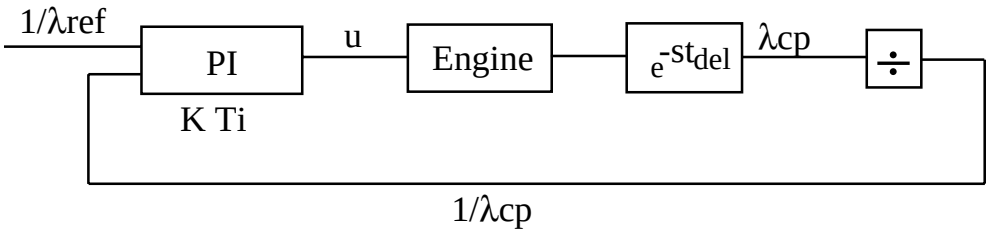

Figure 5.9. Structure of the single UEGO sensor controller.

to the amount of fuel ($1/\lambda = 14.57/(A/F) \approx F/\text{constant}$, for a given throttle angle). A calculated output signal affects all four fuel injectors and no consideration is taken to cylinder individual differences.

The UEGO sensor characteristic, shown in Figure 2.5, is nonlinear but known and invertible. Because it's better to use a linear input to a regulator we linearize the UEGO sensor by applying the inverted function on the sensor signal. This is implemented in the code by a volt2lambda function, see the beginning of Appendix C.

Simulations

By starting with simulations we get a feeling for the process and good starting values for the experimental tests. For a sketch of the simulation model see Appendix B, Figure B.2. To find suitable regulator parameters we apply Ziegler–Nichols rules of thumb. By forcing the process into stable self oscillation we get $K = 0.00405$ and $T_i = 108$. These parameters result in λ_{cp} variations of about ± 3 % around the stoichiometric value. Improved rms values are achieved compared to simulations with the EGO sensor. Note that the settling times (T_1 and T_2) are almost the same as for jump back control. The simulation results are summarized in Table 5.4

Operating point	Reg	λ_{max} cp	λ_{min} cp	λ_{max} sc	λ_{min} sc	T_1	T_2	rms %
1500 stationarity	PI	1.04	0.98	1.22	0.87	—	—	1.50
1500 step	PI	1.22	0.83	1.40	0.75	1.3	1.0	5.28

Table 5.4. Summary of simulation results when using a single UEGO sensor controller.

Implementation

When controlling with the UEGO sensor the pseudo–code differs a little bit from the one used during conventional lambda control. Instead of making a relay interpretation we linearize the filtered input signal using the volt2lambda function. Note that we are now able to set a reference value, in lambda, for the controller to follow. Thus we get:

ReadInput(y);
$y_f = \text{Butterworth}(y)$;
if (control time) **then**
    $\lambda_{cp} = \text{volt2lambda}(y_f)$;
    $e = \frac{1}{ref} - \frac{1}{\lambda_{cp}}$;
    $I = I + \frac{T_s}{T_i}e$;
    $v = Ke + I$;
    if $(v > u_{max})$ **then**
        $u = u_{max}$;
    else if $(v < u_{min})$ **then**
        $u = u_{min}$;
    else
        $u = v$;
    Out(u);
    $I = I + \frac{T_s}{T_t}(u - v)$;
Update filter states;
Wait;

Implementing the pseudo–code above gives us the complete C–code in Appendix C, *void UEGOcontroller*.

Experimental Results

We start at 1500 rpm with no integral part and K set to a decade lower than in simulations, just to be on the safe side. Bringing the system into stable self oscillation by increasing the K value give us Ziegler–Nichols regulator parameters. Since the rules of thumb give an oscillatory behavior of the system we adjust K and T_i to values we consider acceptable. Results of parameter tuning can be viewed in Table 5.5.

Operating point	Reg	K	T_i
1500	PI	0.001 (0.0041)	200 (122)
2250	PI	0.00315 (0.00315)	200 (140)
3000	PI	0.00495 (0.00495)	200 (63)

Table 5.5. Controller parameters used during experiments with a single UEGO sensor controller. Ziegler–Nichols parameters between brackets.

As before we start by showing the stationary behavior at 1500 rpm. Figure 5.10 tells us that we have improved λ_{cp} slightly, now it fluctuates +6 and −3 % around the reference value 1. The output signal from the controller shows neither limit cycles nor jump back phenomena because it's adjusted by new continuous input values of lambda at

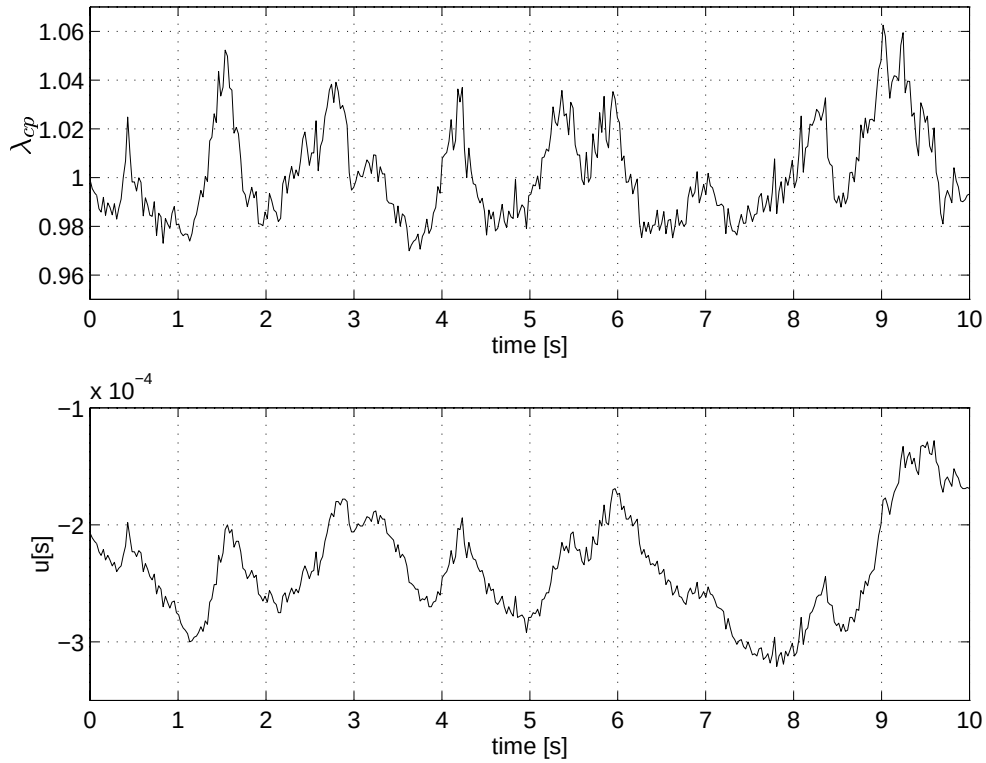

Figure 5.10. Single UEGO sensor controller in stationarity. Top: Lambda at the confluence point. Bottom: Controller output signal.

each sampling interval. In Figure 5.11 we have the step response when making changes in throttle angle position. Notice a lower maximum amplitude compared to Figure 5.6 and Figure 5.8. The settling times (T_1 and T_2) are about half the values achieved by I control using an EGO sensor at low engine speeds. At higher engine speeds the single UEGO controller tends to have longer settling times than the conventional lambda controllers. Although the settling times increase the rms values decrease. For a summary of experimental results study Table 5.6.

Operating point	Reg	λ_{max} cp	λ_{min} cp	λ_{max} sc	λ_{min} sc	T_1	T_2	rms %
1500 stationarity	PI	1.06	0.97	1.21	0.92	—	—	1.91
2250 stationarity	PI	1.03	0.97	1.17	0.97	—	—	1.00
3000 stationarity	PI	1.03	0.98	1.13	0.95	—	—	0.79
1500 step	PI	1.19	0.88	1.38	0.83	1.3	1.4	5.66
2250 step	PI	1.20	0.85	1.42	0.80	2.0	1.6	6.00
3000 step	PI	1.17	0.89	1.31	0.84	2.0	1.3	4.76

Table 5.6. Summary of experimental results when using a single UEGO sensor controller.

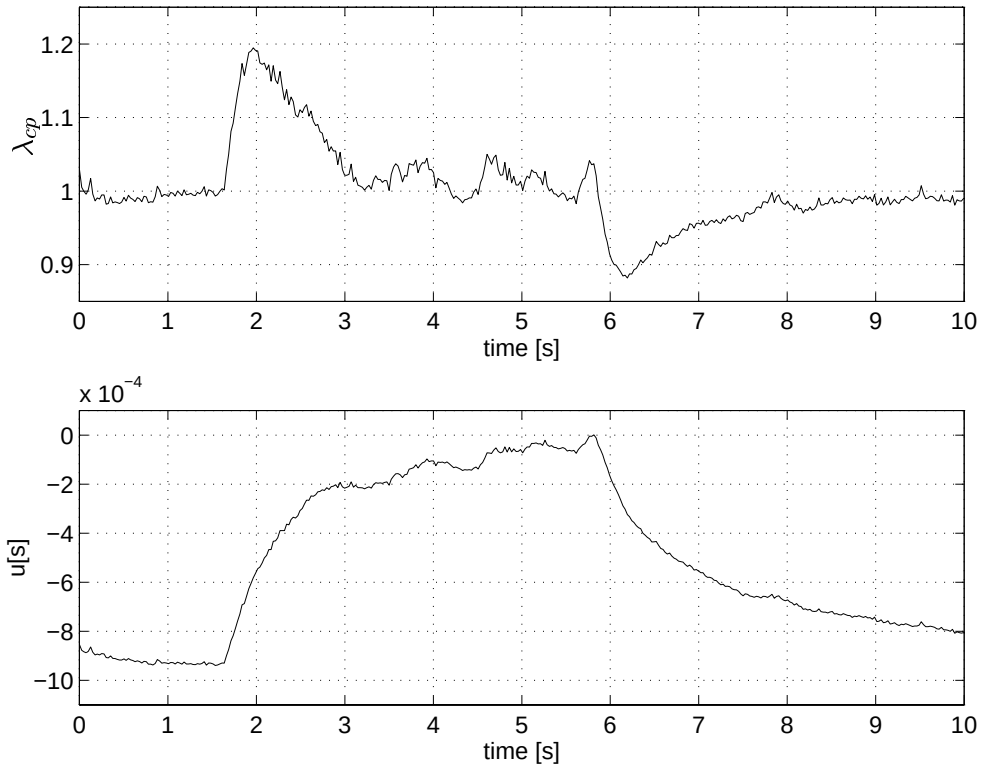

Figure 5.11. Single UEGO sensor controller during transients. Top: Lambda at the confluence point. Bottom: Controller output signal.

5.3 Conclusions

We end this section by making some conclusions regarding conventional and single UEGO sensor lambda control. The statements listed below are based on both simulation experience and experimental results on an actual engine.

- About the same fluctuations at stationarity, ± 5 %.
- Considerably lowered (up to 50 %) rms value when using UEGO control.
- Faster settling times for UEGO sensor control when making steps in throttle angle in most cases.
- Lower maximum deviations from reference value for UEGO control during transients.
- Lower component costs with EGO sensor.
- Some differences in regulator parameters between operating points. For EGO control the K–values differ by a factor 2 and the T_i–values with a factor 3 between the operating points. For UEGO control the K–values differ by a factor 5. Note that for UEGO control the same T_i–value was used at all operating points.
- Using a UEGO sensor gives a possibility to have another reference signal than $\lambda = 1$. This can be utilized during e.g. acceleration and cold starts.
- No possibility to eliminate single cylinder differences with either method.

Almost every above stated argument is, not surprisingly, in favor of the UEGO based control strategy.

6 Multi Sensor Lambda Control

In this section we will show one way to use all information from five lambda sensors to perform cylinder individual lambda control. By a slight modification of the regulator code we also show how the control structure deals with an unknown offset added to measurements of the cylinder individual lambdas. This is done to prepare for an exchange of UEGO measured lambda to lambda estimated by an ionization current A/F algorithm.

Previous we have only used one sensor located slightly downstream the confluence point in the exhaust manifold to perform lambda control. The major benefit of using one sensor for each cylinder is the possibility to eliminate single cylinder differences. Another benefit is shorter transport delay between combustion and measurement. One might wonder if there are any differences between cylinders in an engine? Yes there are! When an engine is new differences are small and may be caused by e.g. non homogenous heat distribution. Differences grow when the engine gets older, due to wear in fuel injectors, inlet valves etc. In [5] the authors talk about differences up to $\pm 5\%$ between single cylinders. Since our test engine is quite new we simulate these differences by adding an, for the regulator, "unknown" offset to each cylinder injection time making some cylinders rich and others lean.

6.1 The Control Structure

This control structure is developed to handle cylinder individual differences in lambda. The structure is common for both the all information regulator and the regulator that has an unknown offset added to its measurements of lambda. As before we use the linearization of UEGO sensors done by the volt2lambda function.

The regulator consists of four inner PI regulators, one for each cylinder, which can be viewed as four separate inner loops, and one outer PI regulator, see Figure 6.1. The inner loops force the cylinder individual lambdas to the same value and the outer loop controls lambda at the confluence point to the desired reference value. The integral (I) part in each regulator is needed for keeping the operating point and we use five integral parts because $\lambda_i = 1$ for $i = 1, \ldots, 4$ may not always result in $\lambda_{cp} = 1$! We also use proportional (P) parts to enhance performance during transients. The reason for not using derivating (D) parts is that the system contains a lot of noise and it's a well known fact that derivating a noisy signal tends to cause problems. Thus we have the same principle control law as before

$$u(t) = Ke(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau.$$

The output signal from the outer regulator is provided as an input signal to the inner loops. The outer regulator uses feedback from the UEGO sensor located downstream the confluence point in the exhaust manifold. This signal together with the reference signal, in lambda, are input signals to the outer regulator. Based on the difference between reference signal and feedback lambda it creates an input signal for the four inner loops. The inner loop regulators use reset anti wind up because they have limitations in their allowed output signals of ± 2 ms.

When having more than one integrating part in a control structure one must consider the fact that the integrating parts can start interacting with each other. It's best understood with an example. Let's form the output signal from the first inner regulator

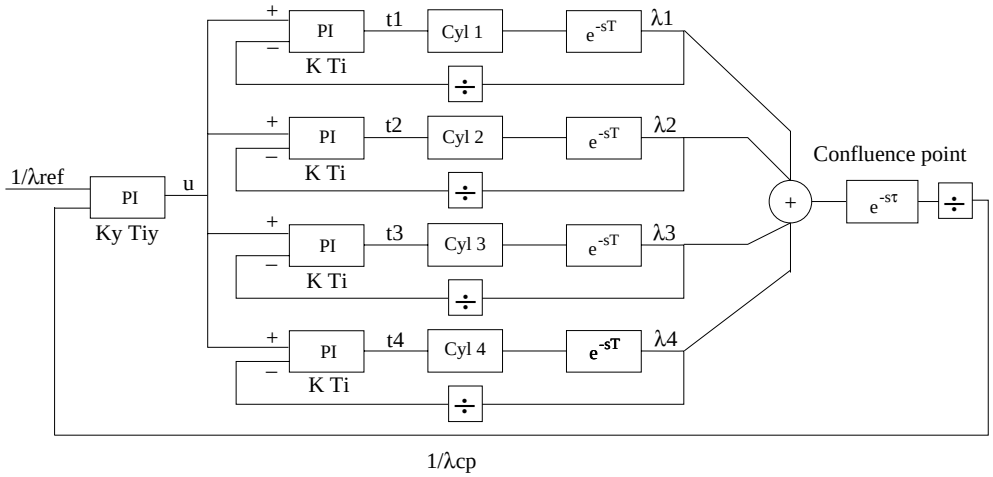

Figure 6.1. Structure of the multi sensor controller.

loop

$$t_1 = K(u - \frac{1}{\lambda_1}) + I_1.$$

Since u is the output signal from the outer regulator we can substitute u and get

$$t_1 = K(K_y(\frac{1}{\lambda_{ref}} - \frac{1}{\lambda_{cp}}) + I - \frac{1}{\lambda_1}) + I_1.$$

Rewritten we have

$$t_1 = KK_y(\frac{1}{\lambda_{ref}} - \frac{1}{\lambda_{cp}}) - \frac{K}{\lambda_1} + KI + I_1.$$

We notice that t_1 depends on the sum of $KI + I_1$ and not on the specific value of I or I_1. Therefore I and I_1 can become very large without affecting the output signal t_1. This isn't a problem in theory, but when implemented on a microcomputer it might cause bad numerical representation and overflow. One can handle this problem by switching integrating parts e.g. when I becomes larger than a specific value a part of I is subtracted from I and added to I_1 via the relationship $KI + I_1 = \text{constant}_1$. The opposite is of course made when I_1 becomes to large[8].

6.2 Implementation

Here we recall the filter choices and sampling rate together with a pseudo–code common for both the all information regulator and the regulator with an unknown offset.

[8] Note that the relationship $KI + I_i = \text{constant}_i$ holds for $i = 1, \ldots, 4$ which means that a change in one integrating part affects the remaining four.

The signals are, like before, filtered through a first order analog low pass filter with a cut–off frequency of 159 Hz. They are than sampled with 2000 Hz and digitally filtered with a second order low pass Butterworth filter with a cut–off frequency of 20 Hz. The regulator works with a frequency of 40 Hz, therefore the filtered signals are decimated at this lower rate of 40 Hz.

The control structure tells us that we need four inner PI controllers and one outer PI controller. The inner loop controllers are not allowed to adjust the basic injection time with more than 2 ms, thus they have limitations in their maximum and minimum output $(t_1, \ldots, t_4)$ of ± 2 ms. In the outer regulator we don't have limitations in the output signal since u is just a reference value for the inner controllers. No limitations mean no need for reset anti wind up. For the same reason as before we let the input signals to the outer regulator be the inverted values of λ_{ref} and λ_{cp}. Considering the input signals to the inner controllers one may wonder why only λ_i is inverted and not u. Let's say that the engine is running lean meaning a $\lambda_{cp} > 1$ resulting in a growing reference value to the inner controllers, u grows. If $t_i = K(u - 1/\lambda_i) + I_i$ a larger u value means larger t_i resulting in enriched air fuel mixture. This is exactly what is desired! Having this in mind we study the pseudo–code given below.

ReadInput($[y, y_1, y_2, y_3, y_4]$);
$[y_f, y_{1f}, y_{2f}, y_{3f}, y_{4f}]$= Butterworth($[y, y_1, y_2, y_3, y_4]$);
if (control time) **then**
$[\lambda_{cp}, \lambda_1, \lambda_2, \lambda_3, \lambda_4]$= volt2lambda($[y_f, y_{1f}, y_{2f}, y_{3f}, y_{4f}]$);

$e = \frac{1}{ref} - \frac{1}{\lambda_{cp}}$;
$I = I + \frac{T_s}{T_{iy}} e$;
$u = K_y e + I$;

$e_1 = u - \frac{1}{\lambda_1}$; /* Controller 1 */
$I_1 = I_1 + \frac{T_s}{T_i} e_1$;
$v_1 = K e_1 + I_1$;
if $(v_1 > t_{max})$ **then**
$t_1 = t_{max}$;
else if $(v_1 < t_{min})$ **then**
$t_1 = t_{min}$;
else
$t_1 = v_1$;
Out(t_1);

$e_2 = u - \frac{1}{\lambda_2}$; /* Controller 2 */
$I_2 = I_2 + \frac{T_s}{T_i} e_2$;
$v_2 = K e_2 + I_2$;
if $(v_2 > t_{max})$ **then**
$t_2 = t_{max}$;
else if $(v_2 < t_{min})$ **then**
$t_2 = t_{min}$;
else
$t_2 = v_2$;

Out(t_2);

$e_3 = u - \frac{1}{\lambda_3}$; /* Controller 3 */
$I_3 = I_3 + \frac{T_s}{T_i} e_3$;
$v_3 = Ke_3 + I_3$;
if $(v_3 > t_{max})$ **then**
    $t_3 = t_{max}$;
else if $(v_3 < t_{min})$ **then**
    $t_3 = t_{min}$;
else
    $t_3 = v_3$;
Out(t_3);

$e_4 = u - \frac{1}{\lambda_4}$; /* Controller 4 */
$I_4 = I_4 + \frac{T_s}{T_i} e_4$;
$v_4 = Ke_4 + I_4$;
if $(v_4 > t_{max})$ **then**
    $t_4 = t_{max}$;
else if $(v_4 < t_{min})$ **then**
    $t_4 = t_{min}$;
else
    $t_4 = v_4$;
Out(t_4);

$I_1 = I_1 + \frac{T_s}{T_t}(t_1 - v_1)$;
$I_2 = I_2 + \frac{T_s}{T_t}(t_2 - v_2)$;
$I_3 = I_3 + \frac{T_s}{T_t}(t_3 - v_3)$;
$I_4 = I_4 + \frac{T_s}{T_t}(t_4 - v_4)$;
Check integral part values and adjust if needed;
Update filter states;
Wait;

Multi argument to Butterworth and volt2lambda means filtering and converting each input signal. The line "Check integral part values and adjust if needed" means that when an integral part becomes to large subtraction of I values are done as explained in Section 6.1. For the complete code see Appendix C, *void lambdacontroller*.

Since the pseudo–code for the controller with unknown offset in lambda is much the same as the code above we explain the extra lines needed instead of repeating the entire pseudo–code when we study the multi sensor controller with offset in the cylinder individual lambdas. We simply expand the pseudo–code by adding an offset estimation line corresponding to Equation 6.1. Since the evaluation of e_i contains λ_i a correction for the offset must be added. Thus we have:

⋮
if (control time) **then**
    ⋮

if (estimation time) **then**
offset estimate $= \frac{\lambda_1+\lambda_2+\lambda_3+\lambda_4}{4} - \lambda_{cp}$;
$\vdots$
$e_1 = u - \frac{1}{\lambda_1 - \text{offset estimate}}$;
$\vdots$

Note that we can control how often an estimation shall occur by setting the estimation time flag. The period between new estimations is determined by how fast and much the offset changes with time. For a discussion on how to choose estimation time see "Estimating the Offset" in Section 6.4.

6.3 A Regulator Using all Available Information

As mentioned above we now utilize all available information from the five UEGO sensors. First we study the results of simulations and later we summarize results of actual engine experiments.

Simulations

Simulations are made to test the new control structure in an easy way. The simulation models for both single cylinder control and multi sensor control can be found in Appendix B, Figure B.3 and Figure B.4. First we need to simulate single cylinder control to find parameters for the four inner loops. Applying Ziegler–Nichols rules of thumb on single cylinder control give $K = 0.009$ and $T_i = 17$. By comparing these values with those for single UEGO sensor control of the complete engine, $K = 0.00405$ and $T_i = 108$, we see that we now have a faster controller. This was expected because the transport delay is now shorter, a fifth of t_{del}. Using the parameters we found for the inner loops and applying Ziegler–Nichols rules of thumb on the outer loop give $K_y = 0.45$ and $T_{iy} = 0.78$. Simulations with the above calculated parameters show a considerable improvement in performance compared to single sensor control. The simulation results are summarized in Table 6.1. We note that the multi sensor controller forces all cylinder lambdas to have a common mean value, equal to 1, as desired. The most remarkable improvement is a dramatic decrease in settling times, from about 1–1.5 s to 0.15 s. This depends mostly on the fact that we now can observe a change in lambda earlier thanks to the shorter transport delay. Another reason is that all cylinders now operate at the same lambda value. We also note that rms during transients is smaller than under stationary conditions for conventional lambda control!

Operating point	Reg	λ_{max} cp	λ_{min} cp	λ_{max} sc	λ_{min} sc	T_1	T_2	rms %
1500 stationarity	PI	1.02	0.99	1.06	0.95	—	—	0.87
1500 step	PI	1.12	0.89	1.15	0.87	0.16	0.14	1.98

Table 6.1. Summary of simulation results when using a multi sensor controller.

Experimental Results

By viewing the control structure we recognize the inner loop controllers to be four single UEGO sensor controllers. If we let the input signal to one single sensor controller be one of the four cylinder individual measured lambdas but still letting the output signal affects all four fuel injectors we are able to find regulator parameters for the inner loop controllers. By applying Ziegler–Nichols rules of thumb we find K and T_i at each operating point. If we switch back to the multi sensor controller structure and use the above found inner loop controller parameters we can tune the outer PI controller parameters, K_y and T_{iy}. The simulations at 1500 rpm resulted in $K_y = 0.45$ and $T_{iy} = 0.78$. Good starting values for engine testing are a 10:th of the K–values given by simulations. Thus letting $K_y = 0.045$ and bringing the system into stable self oscillation we find Ziegler–Nichols controller parameters. Not pleased with the behavior of the system given by these parameters we find some more suitable parameters by manual tuning. A complete list is given in Table 6.2.

Operating point	Reg	K	T_i	K_y	T_{iy}
1500	PI	0.0045 (0.0045)	106 (106)	0.27 (0.54)	2 (1)
2250	PI	0.00315 (0.00315)	111 (111)	0.1 (0.5)	2 (1)
3000	PI	0.00405 (0.00405)	71 (71)	0.3 (0.63)	1 (0.54)

Table 6.2. Controller parameters used during experiments with a multi sensor controller. Ziegler–Nichols parameters between brackets.

To be consistent we show experimental results at 1500 rpm, except for one figure. Studying Figure 6.2 we see (at top) an improved λ_{cp} behavior at stationarity. A ± 2 % fluctuation around the reference value is a large improvement compared to single sensor control and can be explained by better single cylinder performance. From Figure 6.2 we also see that each cylinder has a mean level equal to the reference value, just as expected from simulations. In Figure 6.3 the controllers outputs are illustrated. We see that all outputs show a smooth behavior as desired in stationarity. Especially we note that we can observe the introduced offsets (in fuel injection times) in the inner controllers outputs. Different levels in the outputs are needed to eliminate the offsets.

Comparing the transient response for single sensor and multi sensor control gives the multi sensor control some advantages. Shorter settling times, especially at higher engine speeds with T_1 and T_2 as short as 0.3 to 0.4 s (3000 rpm). Figure 6.4 illustrates how single cylinder differences are eliminated. All four λ_i and λ_{cp} have almost identical appearance. For a summary of engine testing see Table 6.3.

To verify the control properties and performance we study two more cases: regulator start up and change of reference value. Figure 6.5 shows start up. Before the controller is activated the engine is running in open loop with locked fuel map. The confluence point lambda is around 0.96. After approximately four seconds the controller is activated and within a second λ_{cp} stabilizes at the reference value. In Figure 6.5 one also notes how cylinder three is lean before start up and at stoichiometry after controller activation.

To evaluate speed performance we make steps in reference value from rich (ref=0.95) to lean (ref=1.05). The engine response is pictured in Figure 6.6. Engine speed is now 2250 rpm. The change in reference value occurs after 3.5 s resulting in a step response in λ_{cp}.

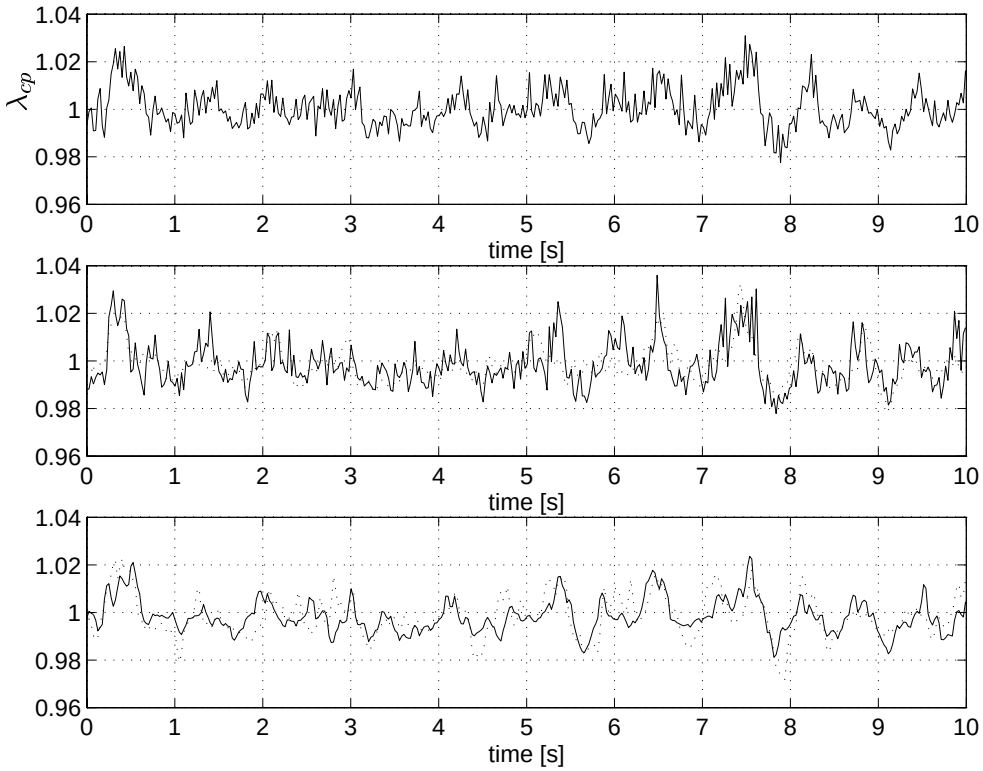

Figure 6.2. Multi sensor controller in stationarity. Top: Lambda at the confluence point. Middle: λ_1 solid and λ_2 dotted. Bottom: λ_3 solid and λ_4 dotted.

Operating point	Reg	λ_{max} cp	λ_{min} cp	λ_{max} sc	λ_{min} sc	T_1	T_2	rms %
1500 stationarity	PI	1.03	0.98	1.04	0.97	—	—	0.84
2250 stationarity	PI	1.03	0.98	1.03	0.98	—	—	0.80
3000 stationarity	PI	1.03	0.98	1.04	0.97	—	—	0.85
1500 step	PI	1.18	0.89	1.16	0.87	0.5	0.8	3.64
2250 step	PI	1.15	0.87	1.17	0.85	0.7	0.8	3.87
3000 step	PI	1.22	0.87	1.24	0.86	0.4	0.3	3.55

Table 6.3. Summary of experimental results when using a multi sensor controller.

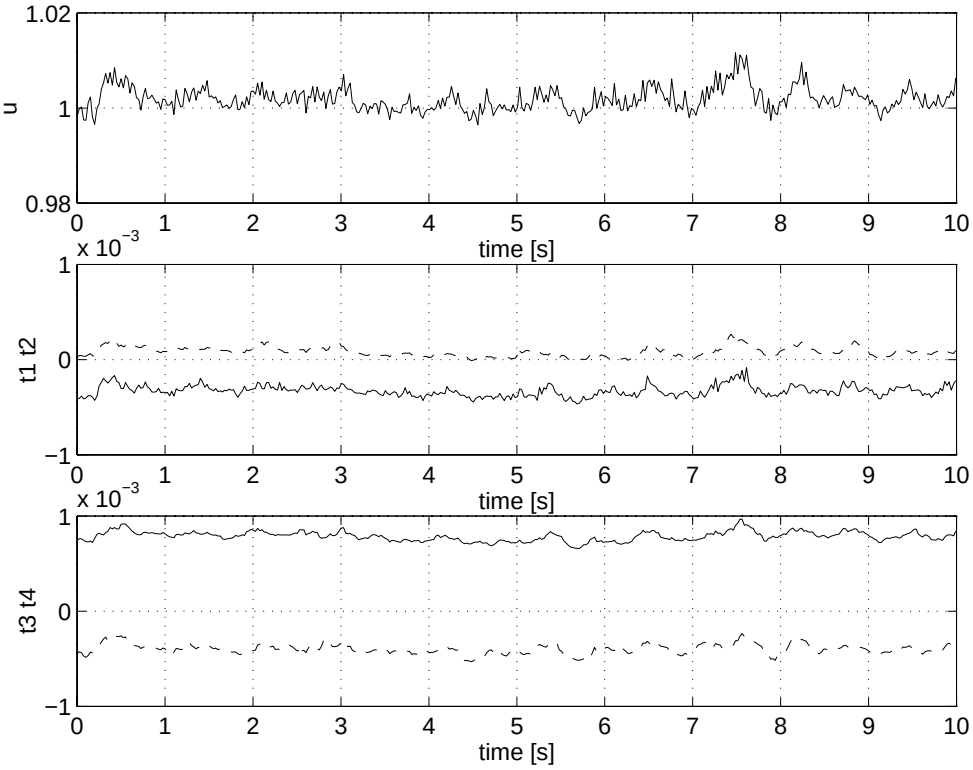

Figure 6.3. Multi sensor controller in stationarity. Top: Output from the outer controller. Middle: t_1 solid and t_2 dashed. Bottom: t_3 solid and t_4 dashed.

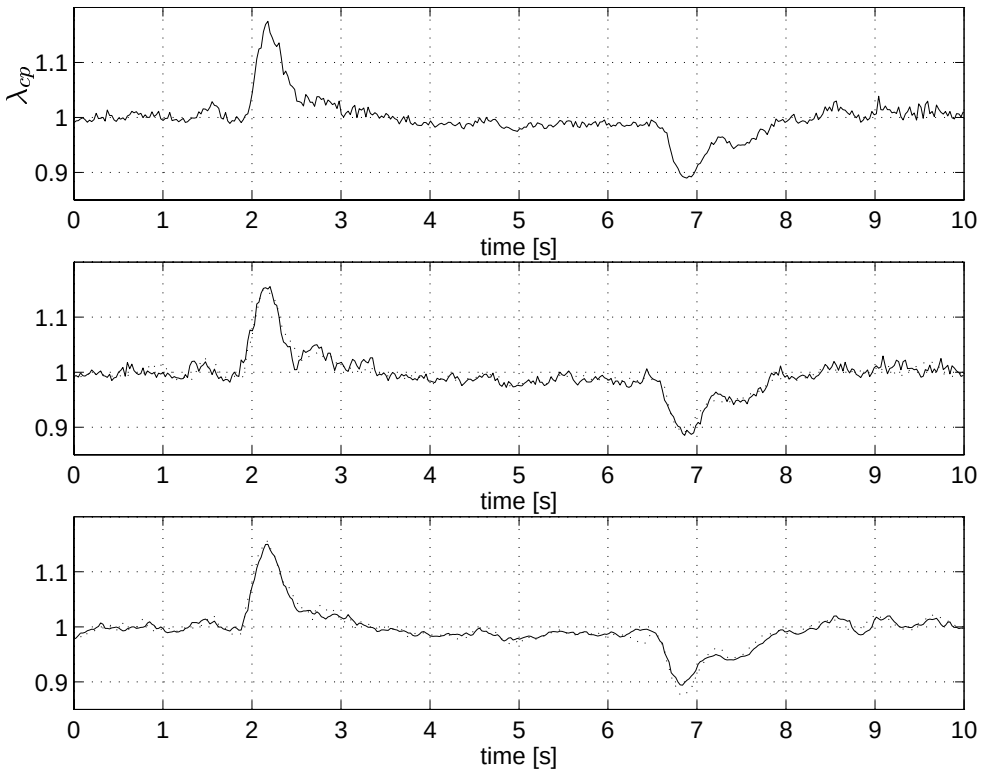

Figure 6.4. Multi sensor controller during transients. Top: Lambda at the confluence point. Middle: λ_1 solid and λ_2 dotted. Bottom: λ_3 solid and λ_4 dotted.

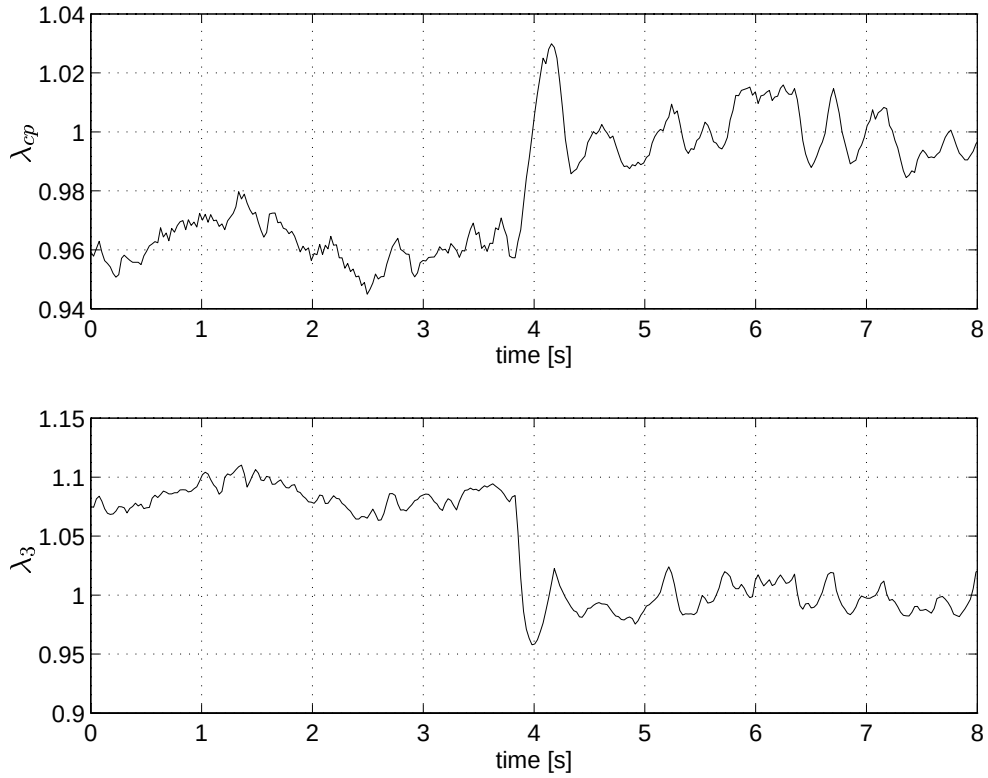

Figure 6.5. Turning on the multi sensor controller. Top: Lambda at the confluence point. Bottom: Lambda in the third cylinder.

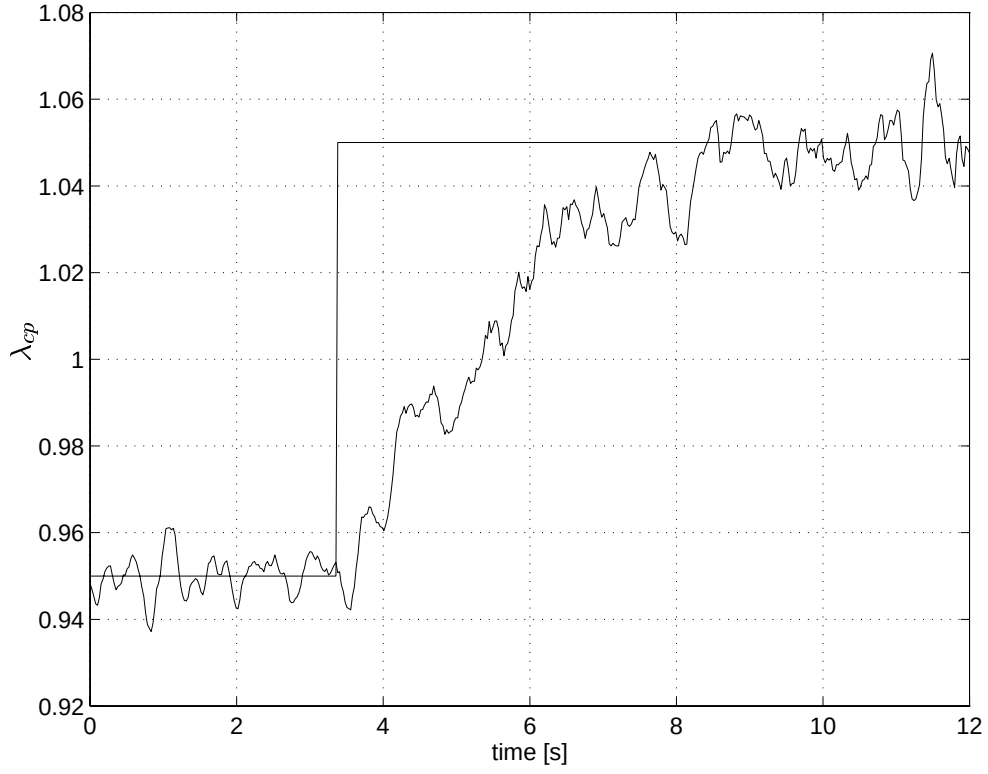

Figure 6.6. Behavior of multi sensor controller when changing reference signal.

6.4 A Regulator with an Unknown Offset in Lambda

A problem occurs when an offset in the cylinder individual lambdas is introduced

$$\lambda_i = \lambda_{i_{true}} + \text{offset} \qquad \text{for } i = 1, \dots, 4.$$

The offset is common for all cylinders. Note that λ_{cp} still is an absolute measurement. In this case it's a disadvantage to use $\frac{1}{\lambda_i}$ instead of λ_i as inputs to the inner loops. It's best understood by studying the following simple example. Let's see what happens with the controller for cylinder 1. For convenience we assume that the reference value to the outer controller is 1. First we study the case without offset. In stationarity u, λ_1 and λ_{cp} are all close to 1. A sudden change in λ_1 from 1 to 0.9 results in

$$e_1 = u - \frac{1}{\lambda_1} = 1 - \frac{1}{0.9} \approx -0.11.$$

Now we assume that the offset is -0.5 i.e. $\lambda_1 = 0.5$ corresponds to stoichiometry. In stationarity u is close to 2, λ_{cp} to 1 and λ_1 to 0.5. A sudden change in λ_1 from 0.5 to 0.4, same change as before, now results in

$$e_1 = u - \frac{1}{\lambda_1} = 2 - \frac{1}{0.4} = -0.5.$$

We note that e_1 depends on the offset which implies that the controller parameters also depend on the offset. This is an undesirable property since the offset changes between operating points. Fortunately there are at least three ways to get rid of the problem.

- Estimate the offset and add it to the measured values.
- Use λ_i instead of $\frac{1}{\lambda_i}$ as inputs to the controllers (similar calculations as above show that this works).
- Use a robust controller.

If we choose to use λ_i instead of $\frac{1}{\lambda_i}$ as inputs we loose the nice property: e being proportional to the fuel injection time. Therefore we choose a controller which estimates the offset and adds it to the cylinder individual lambdas before the e_i's are calculated. Furthermore, if we choose parameter values in such a way that the controller becomes robust a quite coarse approximation of the offset will do.

In the future this controller will pick its measurements of the cylinder individual lambdas from an ionization current algorithm and therefore we first study some preliminary results from this algorithm to get a feeling for the offset. In Table 6.4 values from the algorithm are shown for some operating points and lambdas under stationary conditions. In the table we see that the values decrease when lambda increases. This is undesired from the controllers point of view. To get a more convenient behavior we can for example invert the values or put a minus sign in front of them. If we invert the values a mandatory non linearity is introduced and therefore we choose the latter method. We also see from the table that for a given engine speed, the offset is quite independent of the manifold pressure and lambda value. Furthermore, we see that putting a minus sign in front of the values gives an offset with mean value approximately -1.2 and amplitude 0.1. In the following experiments we let the offset have a period of 5 s. The main reason for choosing this period is that we want to get a proper time scale i.e. a time scale in which we can distinguish offset variations and lambda variations from each other.

speed [rpm]	p_{man} [kPa]	$\lambda = 0.9$	$\lambda = 1.0$	$\lambda = 1.1$
2000	60	0.22	0.13	0.10
2000	wot	0.25	0.15	0.08
4000	60	0.39	0.29	0.19
4000	wot	0.40	0.25	0.15

Table 6.4. Values from the ionization current algorithm, wot means wide open throttle.

Estimating the Offset

First we study what will happen if we have an offset with amplitude 0.1 and period 5 s and don't estimate the offset at all. Note that the offsets mean value in this case must be close to 0 to avoid problems with instability (see above). Figure 6.7 shows that the controller can't eliminate the offset variation at all. From this experiment we see that

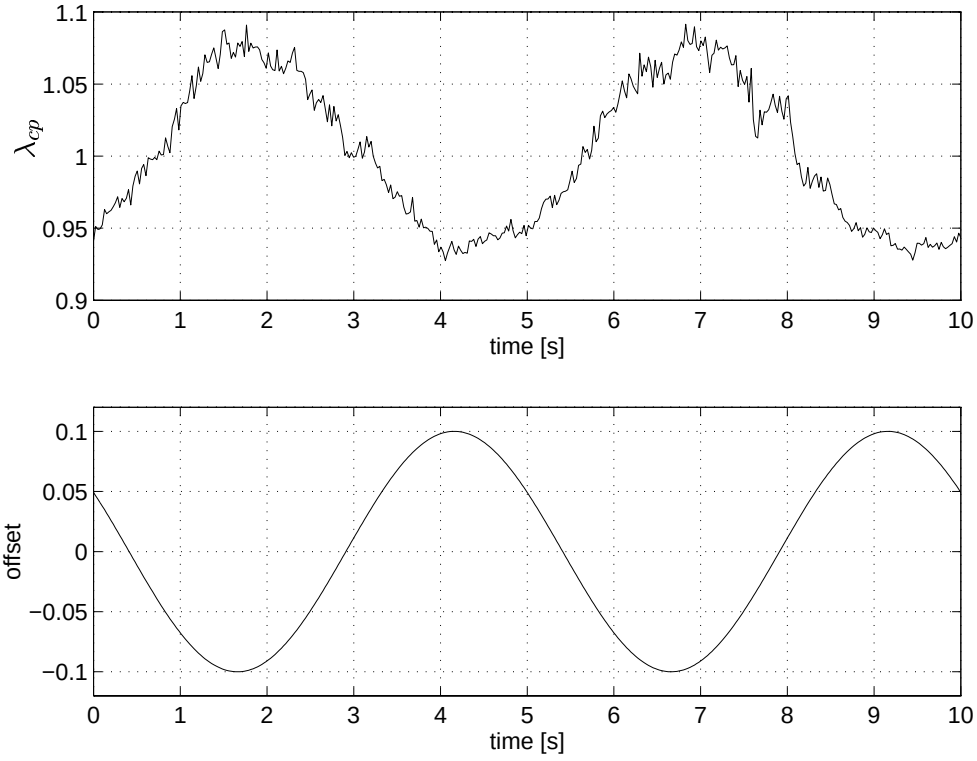

Figure 6.7. Multi sensor controller in stationarity with an offset in the cylinder individual lambdas. No estimation of the offset is done. Top: Lambda at the confluence point. Bottom: Offset in the cylinder individual lambdas.

the controller needs to estimate the offset if it varies in time[9], even if the mean value is close to 0. It's a delicate problem to find a good method to estimate the offset. One has to consider e.g. how often and in which way the estimation shall be done. If we estimate the offset too often the controller will interpret true variations in lambda as offset variations. Since we are only interested in a quite coarse approximation of the

[9] Experience shows that the controller is able to eliminate very slow and small offset variations with mean value close to 0 without estimating the offset.

offset and we have a correct λ_{cp} value we estimate the offset as

$$\text{offset estimate} = \frac{\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4}{4} - \lambda_{cp}. \tag{6.1}$$

Assuming lambda at the confluence point being the mean value of the cylinder individual lambdas isn't theoretically correct, see Section 4.5, but because we only need a quite coarse approximation of the offset it will do here.

Now we try to find some kind of rules telling us how often the estimation shall be done. The offset is from now on

$$\text{offset} = -1.2 + 0.1\sin(2\pi 0.2t). \tag{6.2}$$

Figure 6.8 shows the controllers performance when the controller estimates the offset every 0.2 s i.e. 10 times faster than the offset frequency. We see that this rate for

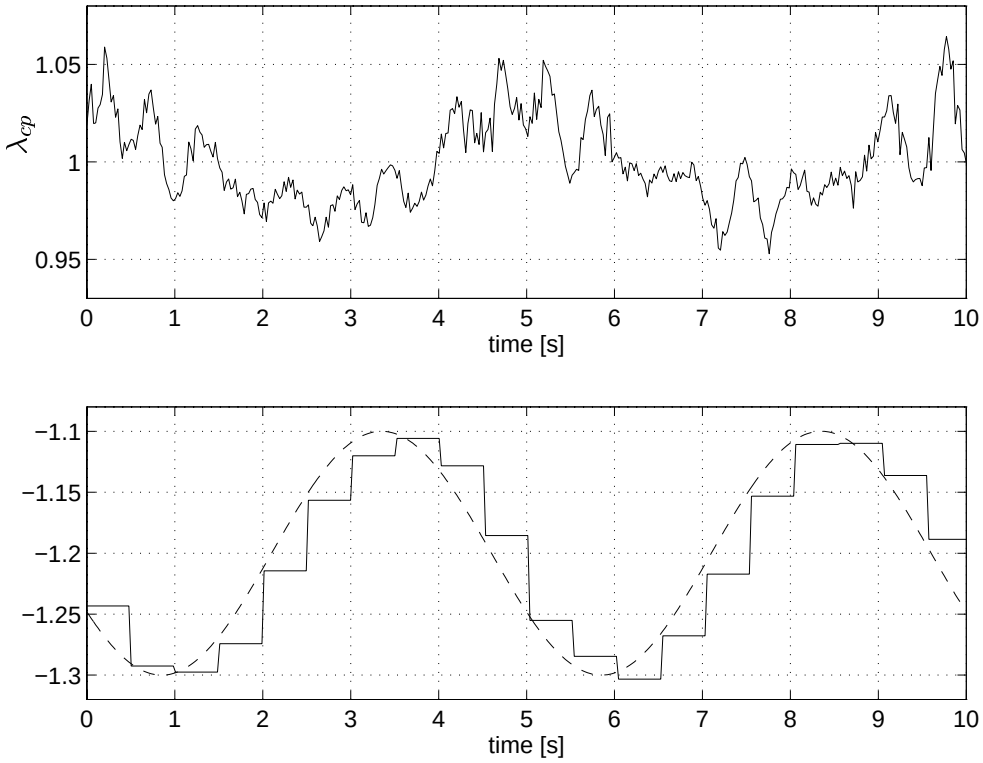

Figure 6.8. Multi sensor controller in stationarity with an offset in the cylinder individual lambdas. Estimation of the offset is done every 0.2 s. Top: Lambda at the confluence point. Bottom: Estimate of the offset, solid, and offset in the cylinder individual lambdas, dashed.

estimating the offset isn't high enough. Further experiments show that the estimating rate shall at least be 50 times the offset variation frequency to get a satisfying behavior in lambda. Of course a higher amplitude in the offset requires a higher estimating rate and vice versa. However, the mean value of the offset doesn't have any influence on the estimating rate at all. Experience shows that if there is much noise in the lambda measurements we can get an even better performance if we let the controller estimate the offset with the controllers sampling rate, 40 Hz. Having this in mind we let the controller use this estimating rate in the following experiments.

Experimental Results

Since the controller nearly eliminates the offsets influence on the control performance we can use the same controller parameter values as when no offset was introduced, see Table 6.2. Because the performance for the controller using all available information and for this controller are much the same, we don't present any figures showing the cylinder individual lambdas and the controllers outputs. Instead we focus on the estimation behavior in stationarity and during transients. However, the experimental results are summarized in Table 6.5.

Operating point	Reg	λ_{max} cp	λ_{min} cp	λ_{max} sc	λ_{min} sc	T_1	T_2	rms %
1500 stationarity	PI	1.03	0.98	1.03	0.98	—	—	0.99
2250 stationarity	PI	1.03	0.98	1.03	0.97	—	—	0.87
3000 stationarity	PI	1.02	0.98	1.03	0.97	—	—	0.70
1500 step	PI	1.15	0.89	1.15	0.88	0.6	0.6	3.74
2250 step	PI	1.18	0.87	1.19	0.85	0.8	0.9	4.25
3000 step	PI	1.19	0.87	1.20	0.84	0.5	0.4	3.53

Table 6.5. Summary of experimental results when using a multi sensor controller and the cylinder individual lambdas containing an offset.

Comparing the results in Table 6.5 with those in Table 6.3 show that this controller has a slight increase in settling times and rms, in most cases. This was expected because the estimate isn't perfect. However, the multi sensor controller is still much better than the single sensor controllers even if an offset is introduced.

Figure 6.9 illustrates the offset estimate under stationary conditions when the engine is running at the operating point 1500 rpm. One notes that the estimate never differs by more than approximately 0.01 from the true value. We also see that the offsets influence on lambda is eliminated as desired.

During transients the controller initially interprets the sudden change in cylinder individual lambdas as a change in offset instead of a true change in lambda, see Figure 6.10 at 2.5 s and 6.75 s. The controller discovers however that the change in cylinder lambda measurements corresponded to a true change when the exhaust gases reach the lambda sensor at the confluence point. From Figure 6.10 we also see that the offset estimate is still close to the true offset.

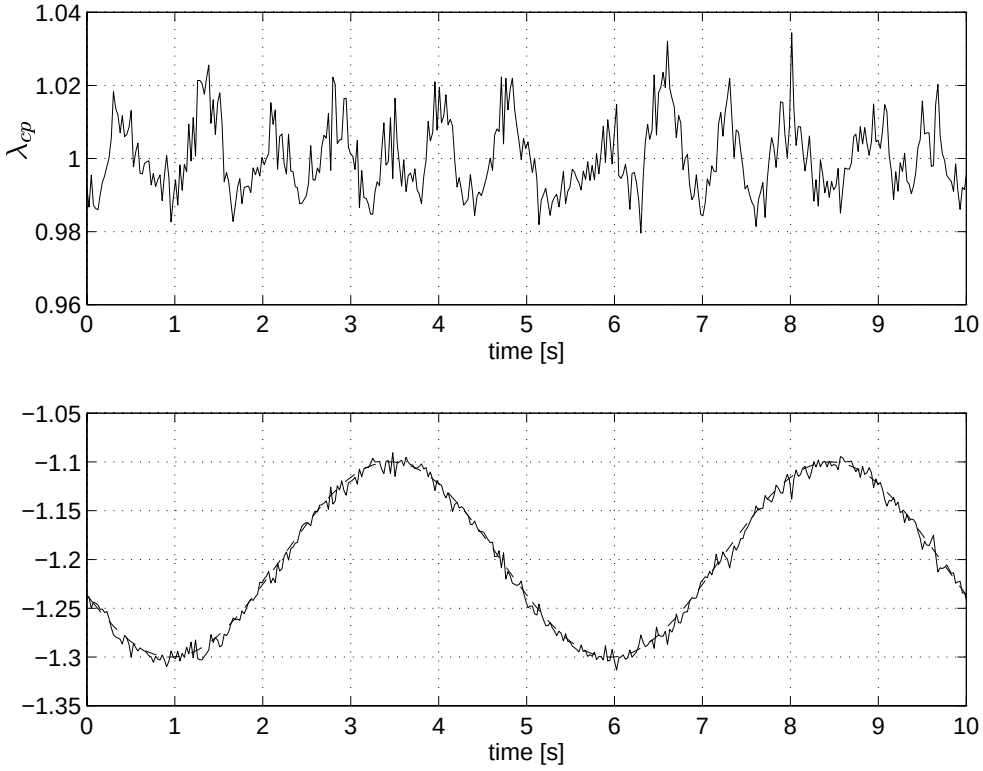

Figure 6.9. Multi sensor controller in stationarity with an offset in the cylinder individual lambdas. Top: Lambda at the confluence point. Bottom: Estimate of the offset, solid, and offset in cylinder the individual lambdas, dashed.

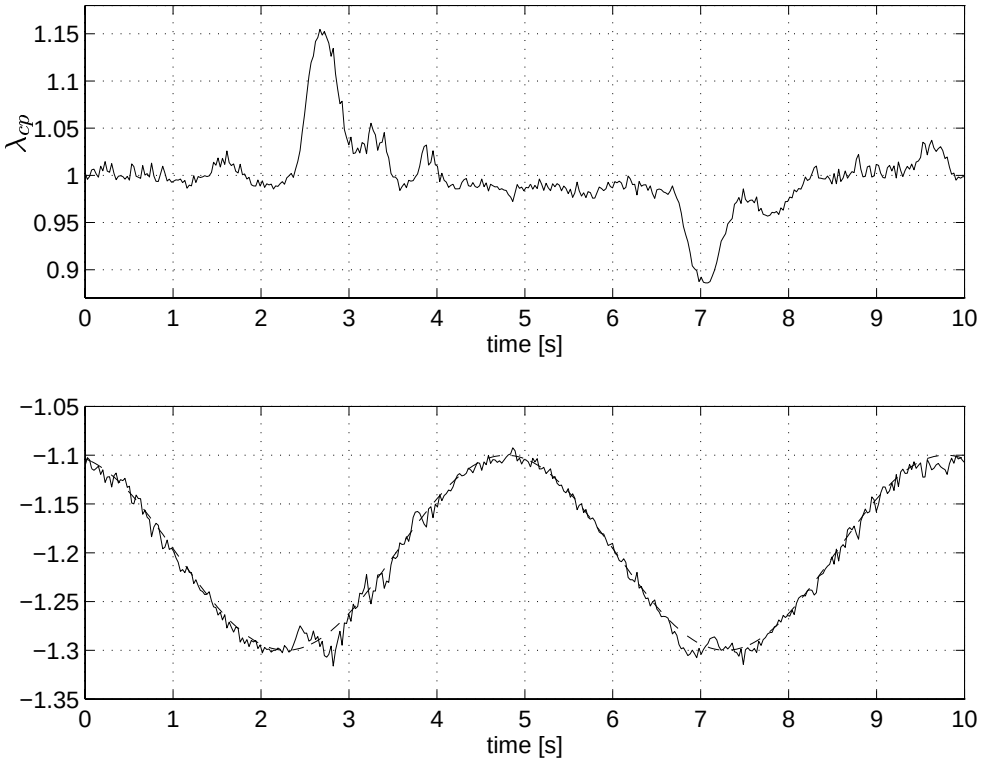

Figure 6.10. Multi sensor controller during transients with an offset in the cylinder individual lambdas. Top: Lambda at the confluence point. Bottom: Estimate of the offset, solid, and offset in the cylinder individual lambdas, dashed.

6.5 Conclusions

By making some conclusions we end this section. The statements listed below hold for both simulations and experimental results.

- Possibility to eliminate single cylinder differences.
- Fluctuations at stationarity of -2 to $+3$ %.
- Low rms values at both stationarity and during transients.
- Short settling times. Especially at higher engine speeds.
- No significant difference in performance between the all information controller and the controller with an unknown sinus offset in the measured cylinder individual lambdas.
- The inner loop controllers use the same Ziegler–Nichols parameters which don't differ much between operating points. Outer controller parameters differ at maximum a factor 3 between operating points.
- A too seldomly estimated offset causes problems and an estimation rate of 50 times the offset frequency will do for an offset amplitude of 0.1.

The conclusion to remember is primarily the ability to perform single cylinder lambda control.

7 Controller Verification

In all previous sections we have simulated an older engine by adding fuel injection time offsets to each cylinder making some cylinders rich and others lean. The additions were made in the regulator code. Results were good and we were able to eliminate these "software" differences between cylinders. To really verify the multi sensor controller we change fuel injectors in two cylinders. Cylinder one has a fuel injector which injects around 10 % more fuel. Cylinder two uses a fuel injector which injects around 18 % less fuel than the standard fuel injectors still used for cylinder three and four. For more details on the fuel injectors see Appendix A. Since the firing order is 1–3–4–2 the modified cylinders are fired after each other. Without a controller the engine performance should be jerky.

Experimental Results

During the engine testing we use no fuel injection time offsets as we have done before. Since the ionization current algorithm for measuring lambda isn't available at this stage we simulate cylinder close lambda measurements with an unknown and varying offset as described in Equation 6.2. We show results from engine experiments at 3000 rpm where we have a basic fuel injection time of about 8.5 ms. Since 20 % of 8.5 ms is approximately 1.9 ms and very close to the maximum output signal (2 ms) allowed from a controller we therefore expand the allowable output range to ± 3 ms.

In Figure 7.1 we see the single cylinder lambdas before and after activation of the controller together with the controllers output signals. Before activation the engine is running open loop with only a basic injection time of about 8.5 ms. After 4.5 s we start the regulator. Within a second the single cylinder lambda values are stabilized at the reference value. Lambda values before controller start up range from $\lambda = 0.85$ to $\lambda = 1.15$. These cylinder individual differences are severe and the engine probably suffers from misfire. The engine sound, during open loop, wouldn't please any person, even with a minimum experience of engine performance. After controller activation the engine runs as with standard fuel injectors.

To verify the differences between standard fuel injectors and non standard we can do as follows. After approximately 5.5 s the controllers output signals are stabilized. By calculating the mean values of the controllers output signals between 5.5 and 10 s we get the controller correction times given in Table 7.1. By adding basic injection time and controller correction time we get the actual injection time, also shown in the table below. The difference between the unmodified cylinders and cylinder 2 is 21 % and for cylinder 1 the difference is 9 %. These values are very close to the values specified by the manufacturer!

Cylinder #	Basic injection	Controller correction	Actual injection
1	8586	-1375	7211
2	8586	1385	9971
3 and 4	8586	-754	7832

Table 7.1. Differences between cylinder injection times when using non standard fuel injectors. All values are in μs.

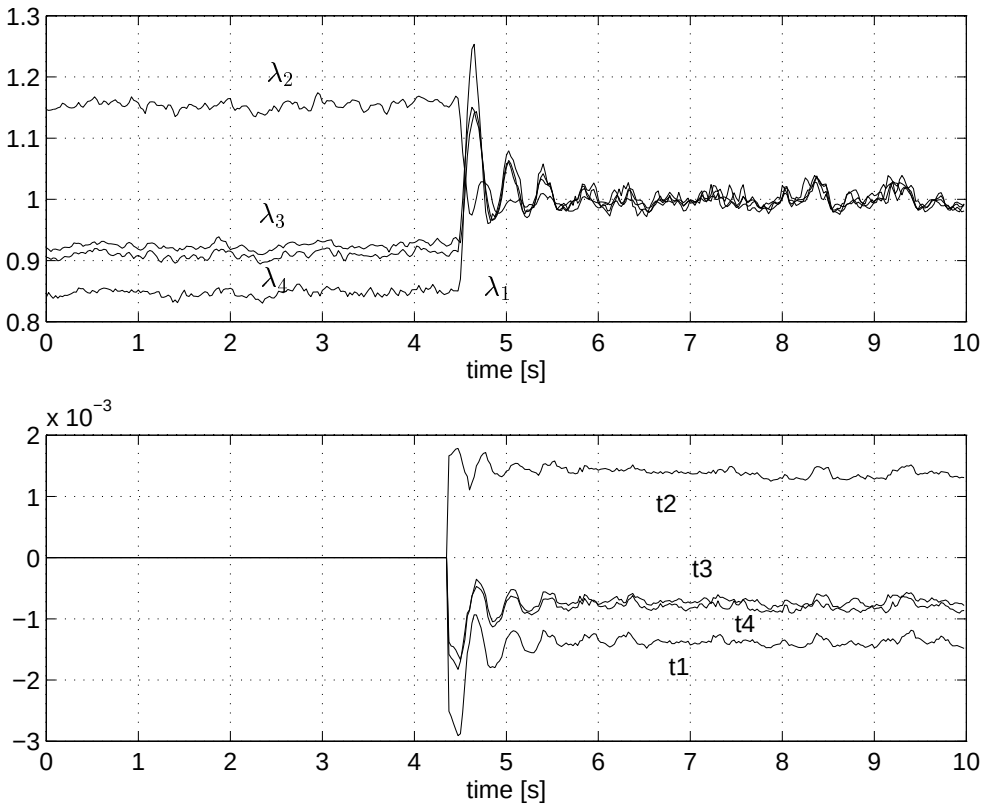

Figure 7.1. Top: Single cylinder lambdas before and after activation of the controller. Bottom: Corresponding inner loop controllers output signals.

Conclusions

The multi sensor controller is able to eliminate differences between cylinders due to different fuel injectors. This means that one can lower the demands of accuracy in the fuel injectors perhaps leading to lower manufacturing costs. Another advantage of single cylinder control is the ability to register too large differences between cylinders and fuel injectors. This can be used for engine diagnosis.

8 Extensions

There are several possible extensions to this work, some of them are presented below. The most interesting extension is an exchange to ionization current based measurements of cylinder individual lambdas. Some problems to solve are how to filter the signals, further studies on how the offset behaves and how to implement the algorithm in an electronic control unit. The results of further offset investigations can be utilized for a more suitable offset estimate. Perhaps with a forgetting factor to prevent interpreting true signal changes as offset variations.

Since the controller parameter values depend on the operating point there is a need for some kind of adaptation for example gain scheduling or parameter adaptive control.

If it's possible to achieve a lower noise level one can introduce derivative parts in the controllers. This might lead to LQG design and other model based controllers. A future engine model probably includes feedforward from throttle angle to achieve better transient response and compensation for sensor dynamics.

Another extension is to introduce learning control i.e. when changing operating point the used values of the I parts are stored in a look up table. Next time this operating point is used the initial I values are picked from the look up table. This gives a possibility to achieve a faster control during transients.

9 Conclusions

The main goal with this project was to perform cylinder individual lambda feedback control to eliminate differences between cylinders. This was achieved with a quite complex, non model based, control structure including four inner parallel PI controllers and one outer PI controller. The results are promising for an exchange to ionization current measured lambdas with an unknown offset. We found that the offset shall be estimated at least 50 times the offset frequency for an offset amplitude of 0.1. A higher amplitude requires a higher estimation rate and vice versa.

The advantages with cylinder individual lambda control are mainly the ability to compensate for differences between cylinders and fuel injectors. This can be used for engine diagnosis, to compensate for wear or lowering the demands on engine components e.g. fuel injectors. Other benefits are a faster lambda control during transients and a more accurate stationarity control. The advantages for multi sensor lambda control with an offset compared with jump back control using an EGO sensor are:

- Between 50 and 60 % enhanced rms values under stationary conditions.
- Between 30 and 60 % enhanced rms values during transients.
- More than halved settling times.
- The ability to set a reference value, in lambda, for the controller to follow.

There are after all some disadvantages with the multi sensor lambda control, namely higher component costs, more complexity and a requirement for more computational power. We also noted a higher noise level in the cylinder close measurements than at the confluence point which is due to turbulence and cycle to cycle variations. The component costs are reduced if ionization current based measurements are used instead of one UEGO sensor per cylinder.

Acceptable controller parameters are given by the Ziegler–Nichols rules of thumb. In the inner loops these parameters are used unchanged and in the outer loop the values are adjusted a factor 5, at the most, towards a slower controller. Good starting values were given by simulations.

Differences in performance between operating points are small but differences in controller parameters exist. For example the maximum difference, between outer controller proportional parts, is a factor 3.

Simulations showed a larger improvement than the experimental results. The improvement is probably due to shorter transport delay, to the cylinder close sensors, than in the actual engine. When using the ionization current algorithm no transport delay exists which implies a possibility of even better lambda control.

References

[1] Simulink, User's Guide, 1992.

[2] G. Almkvist. *Transient Air to Fuel Ratio Response in a Fuel Injected S.I. Engine.* PhD thesis, Chalmers University of Technology, March 1995.

[3] B. A. Ault, V. K. Jones, J. D. Powell, and G. F. Franklin. Adaptive Air–Fuel Ratio Control of a Spark–Ignition Engine. *SAE Paper*, (940373), 1994.

[4] BOSCH. *Automotive Electric/Electronic Systems.* Robert Bosch GmbH, second edition, 1995.

[5] K. J. Bush, N. J. Adams, S. Dua, and C. R. Markyvech. Automatic Control of Cylinder by Cylinder Air–Fuel Mixture Using a Proportional Exhaust Gas Sensor. *SAE Paper*, (940149), 1994.

[6] T. Glad and L. Ljung. *Reglerteknik. Grundläggande teori.* Studentlitteratur, second edition, 1989.

[7] Y. Hasegawa, S. Akazaki, I. Komoriya, H. Maki, Y. Nishimura, and T. Hirota. Individual Cylinder Air–Fuel Ratio Feedback Control Using an Observer. *SAE Paper*, (940376), 1994.

[8] E. Hendricks, M. Jensen, P. Kaidantzis, P. Rasmussen, and T. Vesterholm. Transient A/F Ratio Errors in Conventional SI Engine Controllers. *SAE Paper*, (930856), 1993.

[9] E. Hendricks and S. C. Sorenson. Mean Value Modelling of Spark Ignition Engines. *SAE Paper*, (900616), 1990.

[10] J. B. Heywood. *Internal Combustion Engine Fundamentals.* McGraw-Hill series in mechanical engineering. McGraw-Hill, 1992.

[11] P. Kaidantzis, P. Rasmussen, M. Jensen, T. Vesterholm, and E. Hendricks. Robust, Self-Calibrating Lambda Feedback for SI Engines. *SAE Paper*, (930860), 1993.

[12] L. Ljung and T. Glad. *Modellbygge och simulering.* Studentlitteratur, first edition, 1991.

[13] H. Maki, S. Akazaki, Y. Hasegawa, I. Komoriya, Y. Nishimura, and T. Hirota. Real Time Engine Control Using STR in Feedback System. *SAE Paper*, (950007), 1995.

[14] L. Ljung m.fl. Digital Styrning – kursmaterial, 1995. 4 kompendier.

[15] R. C. Turin and H. G. Geering. Model–Based Adaptive Fuel Control in an SI Engine. *SAE Paper*, (940374), 1994.

Appendix A: Laboratory Facility and Engine Specifications

This appendix will describe the laboratory equipment, at the Division of Vehicular Systems at Linköping University, including engine, dynamometer and computers.

Engine Specifications

The engine is a SAAB 2.3 L spark ignition standard engine equipped with extra sensors for measurement of in cylinder pressure, ionization currents, air mass flow etc. The electronic control unit is called Selma and is developed by Mecel AB. It controls, among many other things, the fuel injectors and spark advance of each cylinder of the engine. Selma is equipped with a CAN–bus[10] interface making it possible to send and receive information during engine tests. The general specifications of the engine are:

Engine type:	4 cylinder, four stroke, 16 valve engine with double overhead camshafts and double balance shafts.
Displacement:	2.3 liters (2290 cm^3).
Bore:	90 mm.
Stroke:	90 mm.
Firing order:	1–3–4–2.
Maximum engine power:	150 bhp (110 kW).
Maximum engine torque:	212 Nm.
Weight:	$\approx$ 160 kg.
Serial number:	B2341.4N10M219569.

Dynamometer

To simulate different driving conditions, engine speed and engine load, we need a dynamometer (brake). There are different types of brakes and the one used in the laboratory is a Dynasyn NT 85 servo motor/generator from Schenck. It can operate under conditions up to engine torques of 150 Nm.

Computers

Besides Selma we use three standard PCs when the engine is running. All three computers and Selma communicate via the CAN–bus, see Figure A.1. The first computer, Hillman, contains a real time system using RTKernel software. It's on this computer our controllers are executed. The injection time offsets calculated by our controllers are send via the CAN–bus to Selma. Hillman also runs a regulator controlling the brake (engine load). From the second computer, Minx, we set reference values for throttle angle and engine speed with help of a graphical user interface with slide bars. On SAAB, the third computer, we can supervise variables used by Selma, for example basic injection time, our calculated injection time offset, actual injection time, engine speed and inlet manifold absolute pressure. From SAAB we are also able to lock/unlock the fuel map and ignition angle.

[10] Controller Area Network

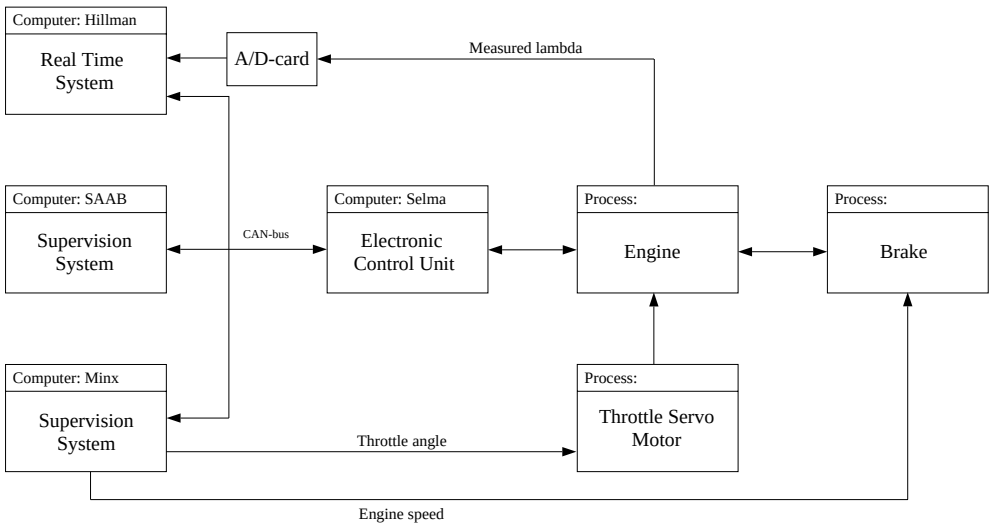

Figure A.1. Hardware setup.

Lambda Sensors

The sensors for measuring lambda are either the standard EGO sensor located 80 cm downstream the exhaust manifold or up to five UEGO sensors of type TL–7111–W1 with electronic controller TC–6000 from NGK. One of the UEGO sensors is located at the same position as the standard EGO sensor. The remaining four are located close to the cylinders measuring cylinder individual lambdas. They are located 12 cm downstream the exhaust manifold. The measured values enter Hillman via a 12 bit A/D conversion card (type RTI–815 from Analog Devices) with a resolution of 4.88 mV.

Fuel Injectors

There are different types of fuel injectors. They differ for example in spray pattern and injected amount of fuel. All together we use three different types of fuel injectors with the same fuel pattern but different injected amount of fuel. They are from BOSCH and are listed below, see Table A.1. The standard fuel injector is type B. The other two

Type	Serial number	cm^3/stroke
A	280150429	224.8
B	280150432	273.4
C	280150418	299.7

Table A.1. Fuel injector types used during engine testing.

are used to verify the regulators ability to eliminate differences between cylinders and fuel injectors. When using type A we get around 18 % less fuel at each injection than normally and with type C around 10 % more. When we in Section 7 use all three types of fuel injectors cylinder 3 and 4 use standard type (B), cylinder 1 uses type C and cylinder 2 type A.

Appendix B: Simulation Models

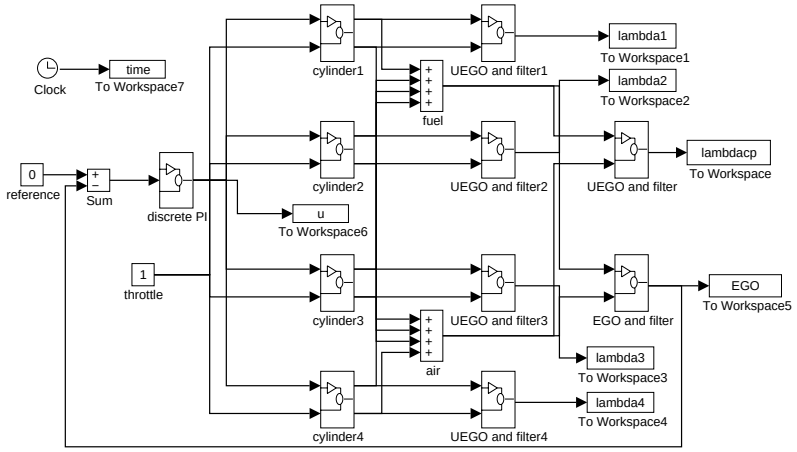

Figure B.1. Simulation model for EGO sensor control.

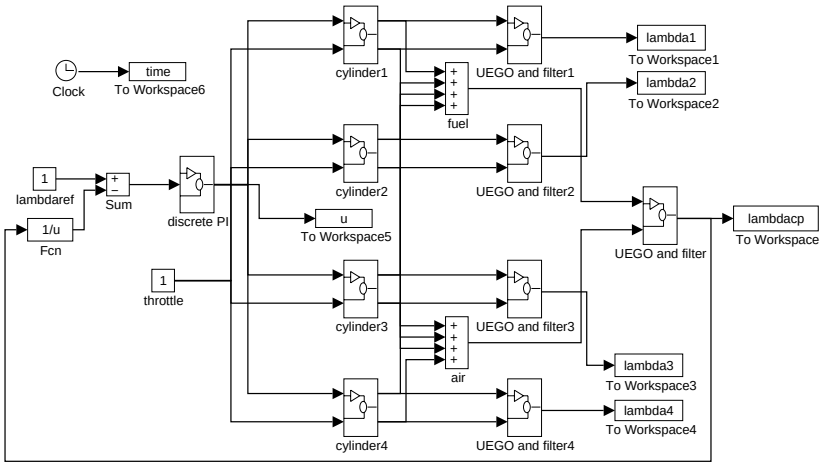

Figure B.2. Simulation model for single UEGO sensor control.

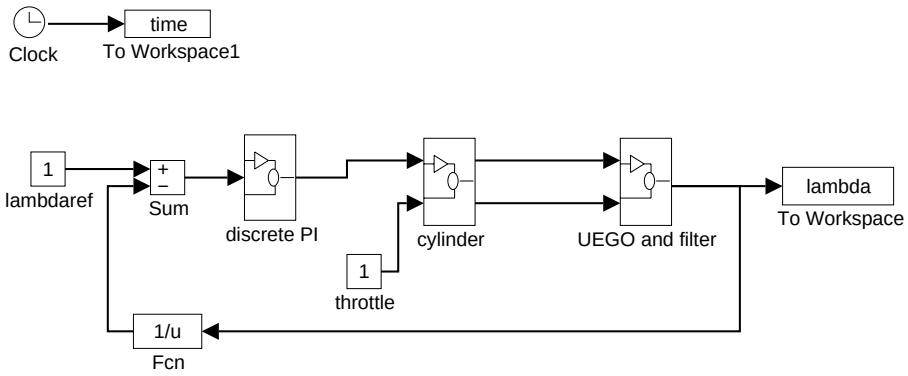

Figure B.3. Simulation model for single cylinder control.

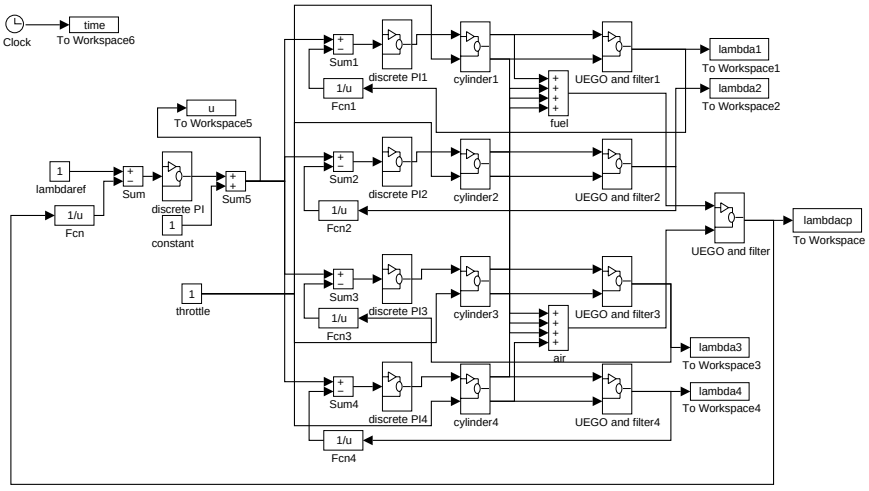

Figure B.4. Simulation model for multi UEGO sensor control.

Appendix C: Code

```
static double volt2lambda(double volt)    /* conversion from UEGO */
{                                         /* sensor signal to lambda */
    double lambda;
    double lambdavector[15] = {9.85, 10.3, 10.75, 11.2, 11.7, 12.15,
                               12.7, 13.25, 13.85, 14.57, 15.95, 17.6,
                               19.75, 22.35, 25.85};
    int i;

    i = floor((volt-1.875)*8);
    if (i<0)
        lambda = lambdavector[0];
    else if (i>=14)
        lambda = lambdavector[14];
    else
        lambda = lambdavector[i] + ((volt - 1.875 - i*0.125)/0.125)*
                 (lambdavector[i+1] - lambdavector[i]);
    lambda = lambda/14.57;    /* normalization */
    return lambda;
}  /* end volt2lambda */
```

```
void EGOcontroller()
{
    Semaphore sem;
    double Ts, Ti, u, to1, to2, to3, to4, K, Tt, lambdacp;
    double umax, umin, ref, e, v, I, lambda1, lambda2, lambda3, lambda4;
    double b[3] = {0.00094469, 0.00188938, 0.00094469}; /* 2:nd order Butter */
    double a[3] = {1, -1.91119706, 0.91497583};         /* fc=20Hz fs=2000Hz */
    double y[3] = {3, 3, 3};
    double y0[3] = {0.5, 0.5, 0.5};
    double y1[3] = {3, 3, 3};
    double y2[3] = {3, 3, 3};
    double y3[3] = {3, 3, 3};
    double y4[3] = {3, 3, 3};
    double yfilt[3] = {3, 3, 3};
    double y0filt[3] = {0.5, 0.5, 0.5};
    double y1filt[3] = {3, 3, 3};
    double y2filt[3] = {3, 3, 3};
    double y3filt[3] = {3, 3, 3};
    double y4filt[3] = {3, 3, 3};
    double *lambdacpvector;
    double *EGOvector;
    double *lambda1vector;
    double *lambda2vector;
    double *lambda3vector;
    double *lambda4vector;
    double *uvector;
    double *timevector;
    int fnum, log, mok, nsamp;
    int i, m, n, ratio;
    char fileName[20];
    unsigned char message[3];
    Port yPort, y0Port, y1Port, y2Port, y3Port, y4Port;
```

```
FILE *file = NULL;
Time nextActivation;

sem = RegisterDouble(Ts);
RegisterDouble(Ti);
RegisterDouble(u);
RegisterDouble(to1);
RegisterDouble(to2);
RegisterDouble(to3);
RegisterDouble(to4);
RegisterDouble(K);
RegisterDouble(Tt);
RegisterDouble(lambdacp);
RegisterInt(fnum);
RegisterInt(log);
RegisterInt(mok);
RegisterInt(nsamp);
RegisterPort(yPort);
RegisterPort(y0Port);
RegisterPort(y1Port);
RegisterPort(y2Port);
RegisterPort(y3Port);
RegisterPort(y4Port);

/*
 *  Default values
 */

Ts = 0.0005;  /* in sec */
Ti = 1e10;
to1 = 0;  /* offset times in microsec */
to2 = 0;
to3 = 0;
to4 = 0;
K = 1;
Tt = Ti;
umax = 0.002;  /* in sec */
umin = -0.002; /* in sec */
ref = 0.5;  /* EGO sensor switch point (in Volt) */
I = 0.0;
lambdacpvector = malloc(500*sizeof(double));
EGOvector = malloc(500*sizeof(double));
lambda1vector = malloc(500*sizeof(double));
lambda2vector = malloc(500*sizeof(double));
lambda3vector = malloc(500*sizeof(double));
lambda4vector = malloc(500*sizeof(double));
uvector = malloc(500*sizeof(double));
timevector = malloc(500*sizeof(double));
fnum = 1;
log = 0;
mok = ((lambdacpvector)&&(EGOvector)&&(lambda1vector)&&
       (lambda2vector)&&(lambda3vector)&&(lambda4vector)&&
       (uvector)&&(timevector));
nsamp = 400;
i = 0;
m = 0;
n = 0;
ratio = 50;  /* controller runs with 1/ratio of 1/Ts */
yPort = -1;
```

```
y0Port = -1;
y1Port = -1;
y2Port = -1;
y3Port = -1;
y4Port = -1;

/*
 *  Wait for start
 */

RTKReceive(NULL, 0);

nextActivation = RTKGetTime();

RTKWait(sem);
while(True) {
    RTKSignal(sem);
    RTKDelayUntil(nextActivation);
    RTKWait(sem);
    nextActivation = RTKGetTime() + Ticks(Ts);

    /* -------- code here ---------------- */

    y[0] = Read(yPort);
    y0[0] = Read(y0Port);
    y1[0] = Read(y1Port);
    y2[0] = Read(y2Port);
    y3[0] = Read(y3Port);
    y4[0] = Read(y4Port);
    yfilt[0] = - a[1]*yfilt[1] - a[2]*yfilt[2] +
               b[0]*y[0] + b[1]*y[1] + b[2]*y[2];
    y0filt[0] = - a[1]*y0filt[1] - a[2]*y0filt[2] +
               b[0]*y0[0] + b[1]*y0[1] + b[2]*y0[2];
    y1filt[0] = - a[1]*y1filt[1] - a[2]*y1filt[2] +
               b[0]*y1[0] + b[1]*y1[1] + b[2]*y1[2];
    y2filt[0] = - a[1]*y2filt[1] - a[2]*y2filt[2] +
               b[0]*y2[0] + b[1]*y2[1] + b[2]*y2[2];
    y3filt[0] = - a[1]*y3filt[1] - a[2]*y3filt[2] +
               b[0]*y3[0] + b[1]*y3[1] + b[2]*y3[2];
    y4filt[0] = - a[1]*y4filt[1] - a[2]*y4filt[2] +
               b[0]*y4[0] + b[1]*y4[1] + b[2]*y4[2];
    i++;

    if (i == ratio) {
        lambdacp = volt2lambda(yfilt[0]);
        lambda1 = volt2lambda(y1filt[0]);
        lambda2 = volt2lambda(y2filt[0]);
        lambda3 = volt2lambda(y3filt[0]);
        lambda4 = volt2lambda(y4filt[0]);

        e = 0.0;
        if ((ref - y0filt[0]) > 0)     /* relay interpretation */
            e = 1;
        else if ((ref - y0filt[0]) < 0)
            e = -1;
        I = I + ratio*Ts*e/Ti;
        v = K*e + I;
        if (v > umax)
            u = umax;
```

```
    else if (v < umin)
        u = umin;
    else
        u = v;

    message[0] = (unsigned char)1;  /* cylinder 1 */
    *((int*)&message[1]) = (int)(u*1000000 + to1); /* in microsec */
    SendCan(15, message);
    message[0] = (unsigned char)2;
    *((int*)&message[1]) = (int)(u*1000000 + to2);
    SendCan(15, message);
    message[0] = (unsigned char)3;
    *((int*)&message[1]) = (int)(u*1000000 + to3);
    SendCan(15, message);
    message[0] = (unsigned char)4;
    *((int*)&message[1]) = (int)(u*1000000 + to4);
    SendCan(15, message);

    I = I + ratio*Ts/Tt*(u - v);

    if ((log == 1)&&(mok)) {
        timevector[n] = SecPerTick*RTKGetTime();
        lambdacpvector[n] = lambdacp;
        EGOvector[n] = y0filt[0];
        lambda1vector[n] = lambda1;
        lambda2vector[n] = lambda2;
        lambda3vector[n] = lambda3;
        lambda4vector[n] = lambda4;
        uvector[n] = u;
        n++;
    }
    if (n == nsamp) {
        sprintf(fileName, "d:EGO%d.txt", fnum);
        file = fopen(fileName, "w");
        fnum++;
        if (file)
            for (m=0;m<nsamp;m++)
                fprintf(file,"%f %f %f %f %f %f %f %f\n",
                        lambdacpvector[m], EGOvector[m],
                        lambda1vector[m], lambda2vector[m],
                        lambda3vector[m], lambda4vector[m],
                        uvector[m], timevector[m]);
        fclose(file);
        file = NULL;
        log = 0;
        n = 0;
    }
    i = 0;
}

y[2] = y[1];  /* update filter states */
y[1] = y[0];
yfilt[2] = yfilt[1];
yfilt[1] = yfilt[0];
y0[2] = y0[1];
y0[1] = y0[0];
y0filt[2] = y0filt[1];
y0filt[1] = y0filt[0];
y1[2] = y1[1];
```

```
        y1[1] = y1[0];
        y1filt[2] = y1filt[1];
        y1filt[1] = y1filt[0];
        y2[2] = y2[1];
        y2[1] = y2[0];
        y2filt[2] = y2filt[1];
        y2filt[1] = y2filt[0];
        y3[2] = y3[1];
        y3[1] = y3[0];
        y3filt[2] = y3filt[1];
        y3filt[1] = y3filt[0];
        y4[2] = y4[1];
        y4[1] = y4[0];
        y4filt[2] = y4filt[1];
        y4filt[1] = y4filt[0];

        /* -------- end code here ------------ */
    }
}  /* end EGOcontroller */



** Init files for EGOcontroller **

make e1(EGOcontroller)
e1.Ti=2000;
e1.K=0.00005;
e1.Tt=2000;
e1.to1=600; operating point 1500
e1.to2=250;
e1.to3=-500;
e1.to4=600;
e1.yPort=4;
e1.y0Port=9;
e1.y1Port=5;
e1.y2Port=6;
e1.y3Port=7;

make e2(EGOcontroller)
e2.Ti=2000;
e2.K=0.00005;
e2.Tt=2000;
e2.to1=500; operating point 2250
e2.to2=250;
e2.to3=-250;
e2.to4=250;
e2.yPort=4;
e2.y0Port=9;
e2.y1Port=5;
e2.y2Port=6;
e2.y3Port=7;
e2.y4Port=8;

make e3(EGOcontroller)
e3.Ti=750;
e3.K=0.0001;
e3.Tt=750;
e3.to1=500; operating point 3000
e3.to2=250;
```

```
e3.to3=-500;
e3.to4=500;
e3.yPort=4;
e3.y0Port=9;
e3.y1Port=5;
e3.y2Port=6;
e3.y3Port=7;
e3.y4Port=8;

** Init files for EGOcontroller **
```

```
void UEGOcontroller()
{
    Semaphore sem;
    double Ts, Ti, u, to1, to2, to3, to4, A, B, f, K, Tt, lambdacp;
    double umax, umin, ref, e, v, I, lambda1, lambda2, lambda3, lambda4;
    double b[3] = {0.00094469, 0.00188938, 0.00094469}; /* 2:nd order Butter */
    double a[3] = {1, -1.91119706, 0.91497583};         /* fc=20Hz fs=2000Hz */
    double y[3] = {3, 3, 3};
    double y1[3] = {3, 3, 3};
    double y2[3] = {3, 3, 3};
    double y3[3] = {3, 3, 3};
    double y4[3] = {3, 3, 3};
    double yfilt[3] = {3, 3, 3};
    double y1filt[3] = {3, 3, 3};
    double y2filt[3] = {3, 3, 3};
    double y3filt[3] = {3, 3, 3};
    double y4filt[3] = {3, 3, 3};
    double *lambdacpvector;
    double *lambda1vector;
    double *lambda2vector;
    double *lambda3vector;
    double *lambda4vector;
    double *refvector;
    double *uvector;
    double *timevector;
    int fnum, log, mok, nsamp;
    int i, m, n, ratio;
    char fileName[20];
    unsigned char message[3];
    Port yPort, y1Port, y2Port, y3Port, y4Port;
    FILE *file = NULL;
    Time nextActivation;

    sem = RegisterDouble(Ts);
    RegisterDouble(Ti);
    RegisterDouble(u);
    RegisterDouble(to1);
    RegisterDouble(to2);
    RegisterDouble(to3);
    RegisterDouble(to4);
    RegisterDouble(A);
    RegisterDouble(B);
    RegisterDouble(f);
    RegisterDouble(K);
    RegisterDouble(Tt);
    RegisterDouble(lambdacp);
    RegisterInt(fnum);
    RegisterInt(log);
    RegisterInt(mok);
    RegisterInt(nsamp);
    RegisterPort(yPort);
    RegisterPort(y1Port);
    RegisterPort(y2Port);
    RegisterPort(y3Port);
    RegisterPort(y4Port);

    /*
     *  Default values
```

```
 */

Ts = 0.0005;  /* in sec */
Ti = 1e10;
to1 = 0;  /* offset times in microsec */
to2 = 0;
to3 = 0;
to4 = 0;
A = 1;  /* default ref = 1 */
B = 0;
f = 0;
K = 1;
Tt = Ti;
umax = 0.002;  /* in sec */
umin = -0.002; /* in sec */
I = 0.0;
lambdacpvector = malloc(500*sizeof(double));
lambda1vector = malloc(500*sizeof(double));
lambda2vector = malloc(500*sizeof(double));
lambda3vector = malloc(500*sizeof(double));
lambda4vector = malloc(500*sizeof(double));
refvector = malloc(500*sizeof(double));
uvector = malloc(500*sizeof(double));
timevector = malloc(500*sizeof(double));
fnum = 1;
log = 0;
mok = ((lambdacpvector)&&(lambda1vector)&&(lambda2vector)&&
       (lambda3vector)&&(lambda4vector)&&(refvector)&&
       (uvector)&&(timevector));
nsamp = 400;
i = 0;
m = 0;
n = 0;
ratio = 50;  /* controller runs with 1/ratio of 1/Ts */
yPort = -1;
y1Port = -1;
y2Port = -1;
y3Port = -1;
y4Port = -1;

/*
 *  Wait for start
 */

RTKReceive(NULL, 0);

nextActivation = RTKGetTime();

RTKWait(sem);
while(True) {
    RTKSignal(sem);
    RTKDelayUntil(nextActivation);
    RTKWait(sem);
    nextActivation = RTKGetTime() + Ticks(Ts);

    /* -------- code here ---------------- */

    y[0] = Read(yPort);
    y1[0] = Read(y1Port);
```

```
y2[0] = Read(y2Port);
y3[0] = Read(y3Port);
y4[0] = Read(y4Port);
yfilt[0] = - a[1]*yfilt[1] - a[2]*yfilt[2] +
           b[0]*y[0] + b[1]*y[1] + b[2]*y[2];
y1filt[0] = - a[1]*y1filt[1] - a[2]*y1filt[2] +
           b[0]*y1[0] + b[1]*y1[1] + b[2]*y1[2];
y2filt[0] = - a[1]*y2filt[1] - a[2]*y2filt[2] +
           b[0]*y2[0] + b[1]*y2[1] + b[2]*y2[2];
y3filt[0] = - a[1]*y3filt[1] - a[2]*y3filt[2] +
           b[0]*y3[0] + b[1]*y3[1] + b[2]*y3[2];
y4filt[0] = - a[1]*y4filt[1] - a[2]*y4filt[2] +
           b[0]*y4[0] + b[1]*y4[1] + b[2]*y4[2];
i++;

if (i == ratio) {
    lambdacp = volt2lambda(yfilt[0]);
    lambda1 = volt2lambda(y1filt[0]);
    lambda2 = volt2lambda(y2filt[0]);
    lambda3 = volt2lambda(y3filt[0]);
    lambda4 = volt2lambda(y4filt[0]);

    ref = A + B*sin(2*3.141593*f*RTKGetTime()*SecPerTick);
    e = (1/ref-1/lambdacp);  /* e proportional to injection time */
    I = I + ratio*Ts*e/Ti;
    v = K*e + I;
    if (v > umax)
        u = umax;
    else if (v < umin)
        u = umin;
    else
        u = v;

    message[0] = (unsigned char)1;  /* cylinder 1 */
    *((int*)&message[1]) = (int)(u*1000000 + to1); /* in microsec */
    SendCan(15, message);
    message[0] = (unsigned char)2;
    *((int*)&message[1]) = (int)(u*1000000 + to2);
    SendCan(15, message);
    message[0] = (unsigned char)3;
    *((int*)&message[1]) = (int)(u*1000000 + to3);
    SendCan(15, message);
    message[0] = (unsigned char)4;
    *((int*)&message[1]) = (int)(u*1000000 + to4);
    SendCan(15, message);

    I = I + ratio*Ts/Tt*(u - v);

    if ((log == 1)&&(mok)) {
        timevector[n] = SecPerTick*RTKGetTime();
        lambdacpvector[n] = lambdacp;
        lambda1vector[n] = lambda1;
        lambda2vector[n] = lambda2;
        lambda3vector[n] = lambda3;
        lambda4vector[n] = lambda4;
        uvector[n] = u;
        refvector[n] = ref;
        n++;
    }
```

```
            if (n == nsamp) {
                sprintf(fileName, "d:UEGO%d.txt", fnum);
                file = fopen(fileName, "w");
                fnum++;
                if (file)
                    for (m=0;m<nsamp;m++)
                        fprintf(file,"%f %f %f %f %f %f %f %f\n",
                                lambdacpvector[m], lambda1vector[m],
                                lambda2vector[m], lambda3vector[m],
                                lambda4vector[m], refvector[m],
                                uvector[m], timevector[m]);
                fclose(file);
                file = NULL;
                log = 0;
                n = 0;
            }
            i = 0;
        }

        y[2] = y[1];  /* update filter states */
        y[1] = y[0];
        yfilt[2] = yfilt[1];
        yfilt[1] = yfilt[0];
        y1[2] = y1[1];
        y1[1] = y1[0];
        y1filt[2] = y1filt[1];
        y1filt[1] = y1filt[0];
        y2[2] = y2[1];
        y2[1] = y2[0];
        y2filt[2] = y2filt[1];
        y2filt[1] = y2filt[0];
        y3[2] = y3[1];
        y3[1] = y3[0];
        y3filt[2] = y3filt[1];
        y3filt[1] = y3filt[0];
        y4[2] = y4[1];
        y4[1] = y4[0];
        y4filt[2] = y4filt[1];
        y4filt[1] = y4filt[0];

        /* -------- end code here ------------ */
    }
}  /* end UEGOcontroller */



** Init files for UEGOcontroller **

make u1(UEGOcontroller)

u1.Ti=200;
u1.K=0.001;
u1.Tt=200;
u1.to1=600; operating point 1500
u1.to2=250;
u1.to3=-500;
u1.to4=600;
u1.yPort=4;
u1.y1Port=5;
```

```
u1.y2Port=6;
u1.y3Port=7;
u1.y4Port=8;

make u2(UEGOcontroller)
u2.Ti=200;
u2.K=0.00315;
u2.Tt=200;
u2.to1=500; operating point 2250
u2.to2=250;
u2.to3=-250;
u2.to4=250;
u2.yPort=4;
u2.y1Port=5;
u2.y2Port=6;
u2.y3Port=7;
u2.y4Port=8;

make u3(UEGOcontroller)
u3.Ti=200;
u3.K=0.00495;
u3.Tt=200;
u3.to1=500; operating point 3000
u3.to2=250;
u3.to3=-500;
u3.to4=500;
u3.yPort=4;
u3.y1Port=5;
u3.y2Port=6;
u3.y3Port=7;
u3.y4Port=8;

** Init files for UEGOcontroller **
```

```
void lambdacontroller()
{
    Semaphore sem;
    double Ts, Tiy, Ti, u, t1, t2, t3, t4, to1, to2, to3, to4, A, B, f, Ky, K,
           Tty, Tt, lambdacp, offsetestimate, offseta, offsetb, offsetf;
    double tmax, tmin, ref, e, e1, e2, e3, e4, v1, v2, v3, v4,
           I, I1, I2, I3, I4, lambda1, lambda2, lambda3, lambda4, offset;
    double b[3] = {0.00094469, 0.00188938, 0.00094469}; /* 2:nd order Butter */
    double a[3] = {1, -1.91119706, 0.91497583};         /* fc=20Hz fs=2000Hz */
    double y[3] = {3, 3, 3};
    double y1[3] = {3, 3, 3};
    double y2[3] = {3, 3, 3};
    double y3[3] = {3, 3, 3};
    double y4[3] = {3, 3, 3};
    double yfilt[3] = {3, 3, 3};
    double y1filt[3] = {3, 3, 3};
    double y2filt[3] = {3, 3, 3};
    double y3filt[3] = {3, 3, 3};
    double y4filt[3] = {3, 3, 3};
    double *lambdacpvector;
    double *lambda1vector;
    double *lambda2vector;
    double *lambda3vector;
    double *lambda4vector;
    double *refvector;
    double *uvector;
    double *t1vector;
    double *t2vector;
    double *t3vector;
    double *t4vector;
    double *timevector;
    double *offsetvector;
    double *offsetestimatevector;
    int fnum, log, mok, nsamp, offsetflag, regflag, eratio;
    int i, m, n, p, ratio;
    char fileName[20];
    unsigned char message[3];
    FILE *file = NULL;
    Port yPort, y1Port, y2Port, y3Port, y4Port;
    Time nextActivation;

    sem = RegisterDouble(Ts);
    RegisterDouble(Tiy);
    RegisterDouble(Ti);
    RegisterDouble(u);
    RegisterDouble(t1);
    RegisterDouble(t2);
    RegisterDouble(t3);
    RegisterDouble(t4);
    RegisterDouble(to1);
    RegisterDouble(to2);
    RegisterDouble(to3);
    RegisterDouble(to4);
    RegisterDouble(A);
    RegisterDouble(B);
    RegisterDouble(f);
    RegisterDouble(Ky);
    RegisterDouble(K);
```

```
RegisterDouble(Tty);
RegisterDouble(Tt);
RegisterDouble(lambdacp);
RegisterDouble(offsetestimate);
RegisterDouble(offseta);
RegisterDouble(offsetb);
RegisterDouble(offsetf);
RegisterInt(fnum);
RegisterInt(log);
RegisterInt(mok);
RegisterInt(nsamp);
RegisterInt(offsetflag);
RegisterInt(regflag);
RegisterInt(eratio);
RegisterPort(yPort);
RegisterPort(y1Port);
RegisterPort(y2Port);
RegisterPort(y3Port);
RegisterPort(y4Port);

/*
 *  Default values
 */

Ts = 0.0005;  /* in sec */
Tiy = 1e10;
Ti = 1e10;
to1 = 0;  /* offset times in microsec */
to2 = 0;
to3 = 0;
to4 = 0;
A = 1;  /* default ref = 1 */
B = 0;
f = 0;
Ky = 1;
K = 1;
Tty = Tiy;
Tt = Ti;
offsetestimate = 0;
offseta = 0;  /* default no offset */
offsetb = 0;
offsetf = 0;
tmax = 0.002;  /* in sec */
tmin = -0.002; /* in sec */
I = 1.0;
I1 = 0.0;
I2 = 0.0;
I3 = 0.0;
I4 = 0.0;
lambdacpvector = malloc(500*sizeof(double));
lambda1vector = malloc(500*sizeof(double));
lambda2vector = malloc(500*sizeof(double));
lambda3vector = malloc(500*sizeof(double));
lambda4vector = malloc(500*sizeof(double));
refvector = malloc(500*sizeof(double));
uvector = malloc(500*sizeof(double));
t1vector = malloc(500*sizeof(double));
t2vector = malloc(500*sizeof(double));
t3vector = malloc(500*sizeof(double));
```

```
t4vector = malloc(500*sizeof(double));
timevector = malloc(500*sizeof(double));
offsetvector = malloc(500*sizeof(double));
offsetestimatevector = malloc(500*sizeof(double));
fnum = 1;
log = 0;
mok = ((lambdacpvector)&&(lambda1vector)&&(lambda2vector)&&
       (lambda3vector)&&(lambda4vector)&&(refvector)&&
       (uvector)&&(t1vector)&&(t2vector)&&(t3vector)&&
       (t4vector)&&(timevector)&&(offsetvector)&&(offsetestimatevector));
nsamp = 400;
offsetflag = 0;
regflag = 1;  /* default controller on */
eratio = 1;  /* default to estimate offset every 1/ratio of 1/Ts */
i = 0;
m = 0;
n = 0;
p = 0;
ratio = 50;  /* controller runs with 1/ratio of 1/Ts */
yPort = -1;
y1Port = -1;
y2Port = -1;
y3Port = -1;
y4Port = -1;

/*
 *  Wait for start
 */

RTKReceive(NULL, 0);

nextActivation = RTKGetTime();

RTKWait(sem);
while(True) {
    RTKSignal(sem);
    RTKDelayUntil(nextActivation);
    RTKWait(sem);
    nextActivation = RTKGetTime() + Ticks(Ts);

    /* -------- code here ---------------- */

    y[0] = Read(yPort);
    y1[0] = Read(y1Port);
    y2[0] = Read(y2Port);
    y3[0] = Read(y3Port);
    y4[0] = Read(y4Port);
    yfilt[0] = - a[1]*yfilt[1] - a[2]*yfilt[2] +
               b[0]*y[0] + b[1]*y[1] + b[2]*y[2];
    y1filt[0] = - a[1]*y1filt[1] - a[2]*y1filt[2] +
               b[0]*y1[0] + b[1]*y1[1] + b[2]*y1[2];
    y2filt[0] = - a[1]*y2filt[1] - a[2]*y2filt[2] +
               b[0]*y2[0] + b[1]*y2[1] + b[2]*y2[2];
    y3filt[0] = - a[1]*y3filt[1] - a[2]*y3filt[2] +
               b[0]*y3[0] + b[1]*y3[1] + b[2]*y3[2];
    y4filt[0] = - a[1]*y4filt[1] - a[2]*y4filt[2] +
               b[0]*y4[0] + b[1]*y4[1] + b[2]*y4[2];
    i++;
```

```
if (i == ratio) {
    lambdacp = volt2lambda(yfilt[0]);
    lambda1 = volt2lambda(y1filt[0]);
    lambda2 = volt2lambda(y2filt[0]);
    lambda3 = volt2lambda(y3filt[0]);
    lambda4 = volt2lambda(y4filt[0]);

    offset = offseta +
             offsetb*sin(2*3.141593*offsetf*RTKGetTime()*SecPerTick);
    if (offsetflag){
        p++;
        if (p >= eratio){
            offsetestimate = ((lambda1 + offset) + (lambda2 + offset)
            + (lambda3 + offset) + (lambda4 + offset))/4 - lambdacp;
            p = 0;
        }
    }

    if (regflag){
        ref = A + B*sin(2*3.141593*f*RTKGetTime()*SecPerTick);
        e = (1/ref-1/lambdacp);
        I = I + ratio*Ts*e/Tiy;
        u = Ky*e + I;

        e1 = (u - 1/(lambda1 + offset - offsetestimate)); /* cyl 1 */
        I1 = I1 + ratio*Ts*e1/Ti;
        v1 = K*e1 + I1;
        if (v1 > tmax)
            t1 = tmax;
        else if (v1 < tmin)
            t1 = tmin;
        else
            t1 = v1;
        message[0] = (unsigned char)1;
        *((int*)&message[1]) = (int)(t1*1000000 + to1);
        SendCan(15, message);                              /* cyl 1 */

        e2 = (u - 1/(lambda2 + offset - offsetestimate)); /* cyl 2 */
        I2 = I2 + ratio*Ts*e2/Ti;
        v2 = K*e2 + I2;
        if (v2 > tmax)
            t2 = tmax;
        else if (v2 < tmin)
            t2 = tmin;
        else
            t2 = v2;
        message[0] = (unsigned char)2;
        *((int*)&message[1]) = (int)(t2*1000000 + to2);
        SendCan(15, message);                              /* cyl 2 */

        e3 = (u - 1/(lambda3 + offset - offsetestimate)); /* cyl 3 */
        I3 = I3 + ratio*Ts*e3/Ti;
        v3 = K*e3 + I3;
        if (v3 > tmax)
            t3 = tmax;
        else if (v3 < tmin)
            t3 = tmin;
        else
            t3 = v3;
```

```
message[0] = (unsigned char)3;
*((int*)&message[1]) = (int)(t3*1000000 + to3);
SendCan(15, message);                                /* cyl 3 */

e4 = (u - 1/(lambda4 + offset - offsetestimate)); /* cyl 4 */
I4 = I4 + ratio*Ts*e4/Ti;
v4 = K*e4 + I4;
if (v4 > tmax)
    t4 = tmax;
else if (v4 < tmin)
    t4 = tmin;
else
    t4 = v4;
message[0] = (unsigned char)4;
*((int*)&message[1]) = (int)(t4*1000000 + to4);
SendCan(15, message);                                /* cyl 4 */

I1 = I1 + ratio*Ts/Tt*(t1 - v1);
I2 = I2 + ratio*Ts/Tt*(t2 - v2);
I3 = I3 + ratio*Ts/Tt*(t3 - v3);
I4 = I4 + ratio*Ts/Tt*(t4 - v4);

if (abs(I)>10){
    I1 = I1 + K*I/2;
    I2 = I2 + K*I/2;
    I3 = I3 + K*I/2;
    I4 = I4 + K*I/2;
    I = I/2;
    printf("I too big\n");
}

if (abs(I1)>10){
    I = I + I1/(2*K);
    I2 = I2 - I1/2;
    I3 = I3 - I1/2;
    I4 = I4 - I1/2;
    I1 = I1/2;
    printf("I1 too big\n");
}

if (abs(I2)>10){
    I = I + I2/(2*K);
    I1 = I1 - I2/2;
    I3 = I3 - I2/2;
    I4 = I4 - I2/2;
    I2 = I2/2;
    printf("I2 too big\n");
}

if (abs(I3)>10){
    I = I + I3/(2*K);
    I1 = I1 - I3/2;
    I2 = I2 - I3/2;
    I4 = I4 - I3/2;
    I3 = I3/2;
    printf("I3 too big\n");
}

if (abs(I4)>10){
```

```
        I = I + I4/(2*K);
        I1 = I1 - I4/2;
        I2 = I2 - I4/2;
        I3 = I3 - I4/2;
        I4 = I4/2;
        printf("I4 too big\n");
    }
}

else {
    message[0] = (unsigned char)1;   /* send only offset times */
    *((int*)&message[1]) = (int)(to1);
    SendCan(15, message);
    message[0] = (unsigned char)2;
    *((int*)&message[1]) = (int)(to2);
    SendCan(15, message);
    message[0] = (unsigned char)3;
    *((int*)&message[1]) = (int)(to3);
    SendCan(15, message);
    message[0] = (unsigned char)4;
    *((int*)&message[1]) = (int)(to4);
    SendCan(15, message);
}

if ((log == 1)&&mok) {
    timevector[n] = SecPerTick*RTKGetTime();
    lambdacpvector[n] = lambdacp;
    lambda1vector[n] = lambda1;
    lambda2vector[n] = lambda2;
    lambda3vector[n] = lambda3;
    lambda4vector[n] = lambda4;
    uvector[n] = u;
    t1vector[n] = t1;
    t2vector[n] = t2;
    t3vector[n] = t3;
    t4vector[n] = t4;
    refvector[n] = ref;
    offsetvector[n] = offset;
    offsetestimatevector[n] = offsetestimate;
    n++;
}

if (n == nsamp) {
    sprintf(fileName, "d:lambda%d.txt", fnum);
    file = fopen(fileName, "w");
    fnum++;
    if (file)
      for (m=0;m<nsamp;m++)
        fprintf(file,"%f %f %f %f %f %f %f %f %f %f %f %f %f %f %f\n",
                    lambdacpvector[m], lambda1vector[m],
                    lambda2vector[m], lambda3vector[m],
                    lambda4vector[m], refvector[m],
                    uvector[m], t1vector[m],
                    t2vector[m], t3vector[m], t4vector[m],
                    timevector[m], offsetvector[m],
                    offsetestimatevector[m]);
    fclose(file);
    file = NULL;
    log = 0;
```

```
                n = 0;
            }
            i = 0;
        }

        y[2] = y[1];  /* update filter states */
        y[1] = y[0];
        yfilt[2] = yfilt[1];
        yfilt[1] = yfilt[0];
        y1[2] = y1[1];
        y1[1] = y1[0];
        y1filt[2] = y1filt[1];
        y1filt[1] = y1filt[0];
        y2[2] = y2[1];
        y2[1] = y2[0];
        y2filt[2] = y2filt[1];
        y2filt[1] = y2filt[0];
        y3[2] = y3[1];
        y3[1] = y3[0];
        y3filt[2] = y3filt[1];
        y3filt[1] = y3filt[0];
        y4[2] = y4[1];
        y4[1] = y4[0];
        y4filt[2] = y4filt[1];
        y4filt[1] = y4filt[0];

        /* -------- end code here ------------ */
      }
}  /* end lambdacontroller */



** Init files for lambdacontroller **

make l1(lambdacontroller)
l1.Ti=106;
l1.Tiy=2;
l1.K=0.0045;
l1.Ky=0.27;
l1.Tt=106;
l1.Tty=2;
l1.to1=600; operating point 1500
l1.to2=250;
l1.to3=-500;
l1.to4=600;
l1.yPort=4;
l1.y1Port=5;
l1.y2Port=6;
l1.y3Port=7;
l1.y4Port=8;

make l2(lambdacontroller)
l2.Ti=111;
l2.Tiy=2;
l2.K=0.00315;
l2.Ky=0.1;
l2.Tt=111;
l2.Tty=2;
l2.to1=500; operating point 2250
```

```
l2.to2=250;
l2.to3=-250;
l2.to4=250;
l2.yPort=4;
l2.y1Port=5;
l2.y2Port=6;
l2.y3Port=7;
l2.y4Port=8;

make l3(lambdacontroller)
l3.Ti=71;
l3.Tiy=1;
l3.K=0.00405;
l3.Ky=0.3;
l3.Tt=71;
l3.Tty=1;
l3.to1=500; operating point 3000
l3.to2=250;
l3.to3=-500;
l3.to4=500;
l3.yPort=4;
l3.y1Port=5;
l3.y2Port=6;
l3.y3Port=7;
l3.y4Port=8;

** Init files for lambdacontroller **
```