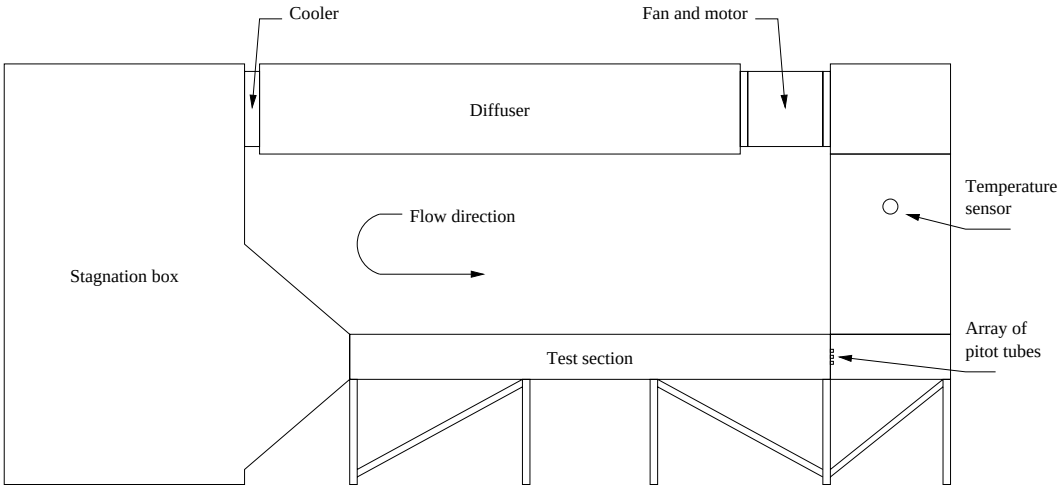
Aalborg University
Institute of Electronic Systems
Department of Control Engineering

Design of a Controller for a Wind Tunnel



6. Semester • Group 631 • 2002

Department of Control Engineering
Aalborg University, Institute of Electronic Systems

TITLE: Design of a Controller for a Wind Tunnel

PROJECT PERIOD: 4. February – 31. May, 2002

PROJECT GROUP: 631

GROUP MEMBERS:

Michael Skipper Andersen
René Just Nielsen
Michael Pedersen
Jørgen Friis
Johnny Jensen
Niels Nørregård Hansen

SUPERVISOR:

Jan Dimon Bendtsen

RUN: 10

MAIN REPORT: 105

APPENDICIES: 36

ENDED: 31. May 2002

ABSTRACT:

This project deals with the construction of a controller for the wind tunnel situated at the Institute of Energy Technology, Aalborg University.
This report documents a theoretical analysis and modeling of the physical wind tunnel together with an experimental modeling of the actuator of the tunnel, consisting of an AC-motor with a fan mounted on the shaft and a frequency converter. Then it deals with the design of three controllers: a PID, a PID with an input filter and a state-space controller. With the use of a performance index and the controller features, the best suited controller for the tunnel is chosen. Further the report documents the design of a graphical user interface for Windows® 2000 that is the interface between the user and the control system. The interface is written in C.
The project has resulted in the control software PelecanWare together with a PID controller implemented in the wind tunnel control system.
Due to a very late instrumentation of the tunnel, it has not been possible to make a model verification. Therefore the controller designs are based on the the unverified models, for which reason the controllers have been coarsely adjusted to be usable in the tunnel system.

Afdeling for Proceskontrol
Aalborg Universitet, Institut for Elektroniske Systemer

TITEL: Design af en regulator til en vindtunnel

PROJEKTPERIODE: 4. februar – 31. maj, 2002

PROJEKTGRUPPE: 631

GRUPPEMEDLEMMER:

Michael Skipper Andersen
René Just Nielsen
Michael Pedersen
Jørgen Friis
Johnny Jensen
Niels Nørregård Hansen

VEJLEDER:

Jan Dimon Bendtsen

OPLAG: 10

SIDER: 105

APPENDIX: 36

AFSLUTTET: 31. maj 2002

SYNOPSIS:

Dette projekt omhandler opbygningen af en regulator og en styrings-/overvågningsenhed til vindtunnelen placeret på Institutit for Energiteknik, Aalborg Universitet.
Denne rapport dokumenterer en teoretisk analyse og modeldannelse af den fysiske vindtunnel samt en eksperimentelt baseret modeldannelse af tunnelens aktuator, bestående af en AC-motor påmonteret en propel og en frekvensomformer. Derefter beskrives designet af tre regulatorer til systemet: en PID, en PID med input-filter og en state-space-regulator. Der er på baggrund af et performance-index og regulatorkarakteristika foretaget et valg af den bedst egnede regulator til tunnelen. Yderligere rummer rapporten designet af en styrings-/overvågningsapplikation til Windows® 2000, programmeret i C. Denne har til formål at fungere som grafisk grænseflade imellem brugeren og tunnelen.
Projektet har resulteret i applikationen PelecanWare samt en PID-regulator, der er blevet realiseret i tunnelens kontrolsystem.
Grundet en meget sen instrumentering af tunnelen har det ikke været muligt at foretage en modelverifikation. Derfor har regulatordesignene været baseret på en ikke-verificeret model, hvorfor de designede regulatorer er justeret kraftig for at gøre dem brugbare i tunnelen.

Preface

This report is written by group 631 at the Department of Control Engineering at Aalborg University in the period from the 4. of February to the 30. of May 2002. The theme for the period is "control technology". The report documents the design of a controller for a wind tunnel situated at the Institute of Energy Technology at Aalborg University.

The reader of the report is assumed to have a knowledge corresponding to that of the sixth semester control engineering or above.

The references in the text are done by chapter or appendix number and section number. An example of a reference to chapter 2 section 1 is section 2.1.

A reference to a figure is done by number of chapter or appendix and the consecutive number of the figure. For example, a reference to the second figure in appendix B would be (figure B.2). The same goes for references to equations. An example of a reference to the fourth equation in chapter 4 is equation 4.4.

A reference to a source is put in angular brackets, e.g. [ogata]. In the bibliography a list with explanations to the references is found.

In the text the abbreviation VLT is used for the frequency converter.

Enclosed in the report is a CDROM containing data sheets, schematics, source code and the like.

Contents

Chapter 1

Introduction

In projects concerning the aerodynamics of a construction, such as tall buildings, bridges and aircrafts e.g., it is often necessary to build small-scale mock-ups of the construction in order to predict the aerodynamic behaviour of the full-scale construction.

Attempts to set up a complete mathematical model of the system, or just parts of it, often led to inappropriately complicated models that are too complex to be of any use. In the case of the construction of a building, for instance, often it is only desirable to know the bottom line outcome of the influence of air velocity on the building, and not the actions of each molecule in the entire system.

In the former case a wind tunnel is a suitable mean for testing aerodynamical behaviours on a small-scale mock-up of the building. With the use of a technique called Laser Doppler Anemometry (LDA) e.g. the air velocity around the mock-up can be registred and analyzed.

The Department of Thermal Energy Technology at the Institute of Energy Technology, Aalborg University has had several different wind tunnels used for research.

The first prototype had a small test section and was a single-pass, open ended wind tunnel which was very sensitive to its surroundings. This had the disadvantage that even small fluctuations in the air pressure outside the wind tunnel, such as a door being opened or closed, would reflect on the measurements and, hence, the prototype was not suitable for research.

Later a student project was established with the purpose of building a closed-loop wind tunnel. It was still small and the air velocity in the test section was not quite stable.

Then, with the experience from the previous wind tunnels and wind tunnels from other test facilities in the world, a new and larger wind tunnel was built with funds from private companies and Aalborg University. With a longer test section and with the introduction of a stagnation box, in which the air velocity can be brought down to zero and reaccelerated, the flow stability in the test section has improved and better results can be obtained.

Flow measurements in the new tunnel are carried out using LDA. By adding coloured dust to the air in the tunnel and projecting a plane of coherent light from a laser through the test section, the light reflections from the dust particles can be registered with a camera and analyzed. Since this type of measurement requires a constant air temperature, a regulated cooler has been installed in the wind tunnel to compensate for the motor's heating of the air.

The air velocity in the wind tunnel is adjusted by manually entering a frequency on the motor's frequency converter and subsequently measuring the air velocity in the test section with a hand held gauge. This, however, means that the velocity in the test section cannot be properly regulated and therefore this student project at the Department of Control Engineering has been established in order to design and implement a controller for the wind tunnel.

At the present moment the wind tunnel is in the process of being equipped with fixed pressure transducers and a temperature transducer. Until these have been properly installed, air velocity and air temperature must be measured with hand-held gauges, which complicates the measuring procedure somewhat.

Chapter 2

Analysis

To be able to design a controller for the wind tunnel a suitable model of this must be set up. This is done in the analysis part where also the various transducers and actuators used in the wind tunnel are analyzed.

A draft of the wind tunnel is shown in figure 2.1.

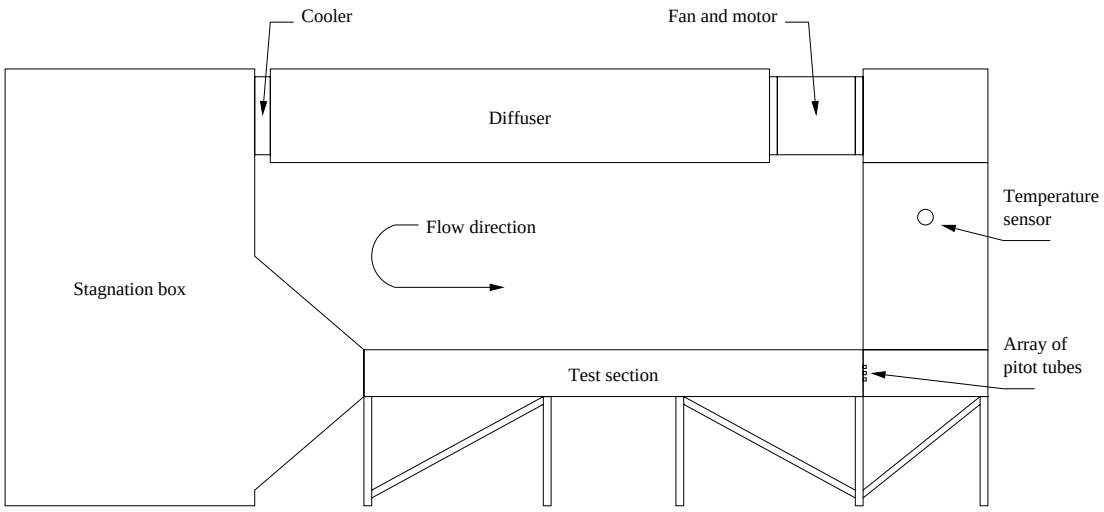


Figure 2.1: A sketch of the wind tunnel.

It consists of the following components:

Fan and Motor: The fan and motor create a pressure difference which causes the air in the tunnel to flow. These two components will be considered in section 2.2.

Diffuser: The purpose of the diffuser is to reduce the velocity of the air from the fan and motor into the stagnation box.

Cooler: To keep the air temperature in the wind tunnel at a constant value a cooler is inserted to compensate for the heating of the air caused by the motor.

Stagnation Box: The stagnation box acts like a buffer and a capacitor that equalizes fluctuations in the air velocity. The air velocity is assumed to be zero in the center of the box, hence the name stagnation box.

Test Section: The test section is a 4 meter duct in which the object to be tested is placed. By means of Laser Doppler Anemometry the air velocity in the test section can be measured.

Pressure Transducers: At the end of the test section an array of 3×3 pitot tubes, connected to six pressure transducers, is mounted. These will be further described in section 2.6.

Temperatur Sensor: The temperature sensor is used only to produce read-outs of the current temperature in the test section to the user.

2.1 Model of the Wind Tunnel

An air velocity occurs only when a difference in air pressure between two points exists. The air velocity in the wind tunnel is determined by the angular velocity of the fan, mounted on the motor shaft. Increasing the angular velocity causes the air pressure in front of the fan to rise and, hence, causes the air to flow.

In order to model the air velocity in the wind tunnel the pressure losses in the wind tunnel must be calculated. Table 2.1 shows the symbols used in the following calculations.

The elements in the wind tunnel which mainly cause air pressure losses are the diffuser, the air cooler, the stagnation box, the test section and the two 90° bends after the test section.

In figure 2.2 these elements are considered in an equivalent diagram of the wind tunnel, where the resistances illustrate the pressure loss factors of the different components of the wind tunnel, and the point a is where the flow rate is measured. From this equivalent diagram the volumen flow can be calculated.

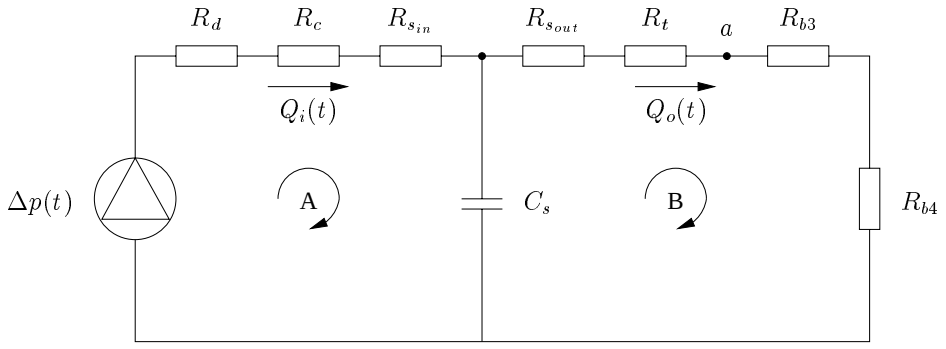


Figure 2.2: Equivalent diagram of the wind tunnel.

In general the pressure loss $\Delta p(t)$ of a specific tunnel element depends on the square of the air velocity $v(t)$ and is given as:

$$\Delta p(t) = \zeta \frac{\rho}{2} v(t)^2 = \zeta \frac{\rho}{2} \left(\frac{Q(t)}{A} \right)^2 = R \cdot Q(t)^2 \tag{2.1}$$

where ζ is the pressure loss factor, ρ is the density of air, A is the area of the tunnel element and $Q(t)$ is the volume flow.

Symbol	Unit	Explanation
$\Delta p(t)$	[Pa]	Pressure increase delivered by the fan and the motor.
ρ	[kg/m^3]	Density of the air.
ζ	[$\cdot$]	Pressure loss factor.
$Q(t)$	[m^3/s]	Volume flow.
A	[m^2]	Area.
C_s	[mol$\cdot$m^3/J]	Capacitance of the stagnation box.
R_{air}	[J/mol/K]	Gas constant.
λ_t	[$\cdot$]	Friction factor related to pressure loss in a duct.
D_{H_t}	[m]	Hydraulic diameter of a duct.
k	[m]	Roughness of duct material.

Table 2.1: Symbols used in the wind tunnel model.

Using Kirchhoff's Voltage Law on figure 2.2 gives the following:

Pressure loss in loop A:

$$\Delta p(t) = \Delta p_d(t) + \Delta p_c(t) + \Delta p_{s_{\text{in}}}(t) + \Delta p_{C_s}(t) \tag{2.2}$$

Pressure loss in loop B:

$$\Delta p_{C_s}(t) = \Delta p_{s_{\text{out}}}(t) + \Delta p_t(t) + \Delta p_{b3}(t) + \Delta p_{b4}(t) \tag{2.3}$$

The Diffuser

A diffuser is a duct with the shape shown on figure 2.3.

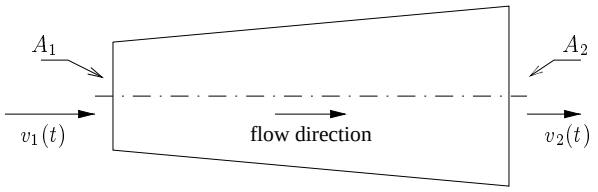


Figure 2.3: A draft of a diffuser.

Since the volume flow into the diffuser ($Q_1(t)$) is equal to the volume flow out of the diffuser ($Q_2(t)$), it follows that:

$$Q_1(t) = Q_2(t) \quad \Leftrightarrow \quad A_1 \cdot v_1(t) = A_2 \cdot v_2(t) \tag{2.4}$$

and therefore the effect of a diffuser is to reduce the air velocity.

Since the diffuser is divided into tree parts, it is considered to be made of three parallel diffusers with the same geometry. Hereby the volumen flow through each of the diffusers will be the same. The pressure loss over one of the diffusers is [pukkila, p 24]:

$$\Delta p_{\text{one diff}}(t) = \zeta_d \frac{\rho}{2} \left(\frac{1}{3} v(t) \right)^2 = \zeta_d \frac{\rho}{2} \left(\frac{1}{3} \cdot \frac{Q_i(t)}{A_d} \right)^2 \tag{2.5}$$

where $v(t)$ is the air velocity and $Q_i(t)$ is the volume flow into the diffuser.

The loss factor ζ_d depends on the ratio between the maximum and minimum area, and A_d is the smallest area of the diffuser, i.e. where the air velocity is highest.

The Cooler

The pressure loss over the cooler is also a function of the square of the velocity:

$$\Delta p_c(t) = \zeta_c \frac{\rho}{2} \left(\frac{Q_i(t)}{A_c} \right)^2 \tag{2.6}$$

where ζ_c is a loss coefficient for the specific cooler.

The Stagnation Box

The stagnation box is considered to consist of a capacitance and an input resistance and an output resistance [ogata]. The pressure losses over the input and output resistances are:

$$\Delta p_{s_{\text{in}}}(t) = \zeta_{s_{\text{in}}} \frac{\rho}{2} \left(\frac{Q_i(t)}{A_{s_{\text{in}}}} \right)^2 \tag{2.7}$$

$$\Delta p_{s_{\text{out}}}(t) = \zeta_{s_{\text{out}}} \frac{\rho}{2} \left(\frac{Q_o(t)}{A_{s_{\text{out}}}} \right)^2 \tag{2.8}$$

The pressure loss over the capacitance is:

$$\Delta p_{C_s}(t) = \frac{1}{C_s} \int_{-\infty}^{t} Q_i(\tau) - Q_o(\tau) d\tau \tag{2.9}$$

The capacitance C_s is given as [ogata, p 193]:

$$C_s = \frac{V_{C_s}}{\gamma \cdot R_{\text{air}} \cdot T} \tag{2.10}$$

where V_{C_s} is the volume of the stagnation box and R_{air} is 287 $\text{J/K} \cdot \text{mol}$. γ is the ratio between the specific heats $\frac{c_v}{c_p}$, where c_v is the specific heat at constant volume and c_p is the specific heat at constant pressure. The value of γ is typically between 1 and 1.2, and can therefore be considered approximately constant.

The Test Section

Due to the length of the test section the pressure loss can be calculated as the pressure loss of a fully developed turbulent flow in a duct [pukkila, p 22]:

$$\Delta p_t(t) = \frac{l_t \lambda_t}{D_{H_t}} \cdot \frac{\rho}{2} \left(\frac{Q_o(t)}{A_t} \right)^2 \tag{2.11}$$

where:

$$\lambda_t = \frac{1}{\left(1.14 - 2 \cdot log \frac{k}{D_{H_t}}\right)^2} \tag{2.12}$$

k is the roughness of the duct material and D_{H_t} is the hydraulic diameter of the test section given by:

$$D_{H_t} = \frac{2ab}{a+b} \tag{2.13}$$

where a and b are the length and width of a cross section of the duct, respectively.

The two 90° Bends

The pressure loss of a 90° bend is given by:

$$\Delta p_{90^\circ}(t) = \zeta_{90^\circ} \frac{\rho}{2} \left(\frac{Q_o(t)}{A_{90^\circ}} \right)^2 \tag{2.14}$$

Thus, the total loss of the two bends is:

$$\Delta p_b(t) = \zeta_{90^\circ} \frac{\rho}{2} \left(\left(\frac{Q_o(t)}{A_{b3}} \right)^2 + \left(\frac{Q_o(t)}{A_{b4}} \right)^2 \right) \tag{2.15}$$

Solving the Equations

Now, equation 2.2 can be further expanded:

$$\begin{aligned} \Delta p(t) &= 3 \cdot \zeta_d \frac{\rho}{2} \left(\frac{1}{3} \cdot \frac{Q_i(t)}{A_d} \right)^2 + \zeta_c \frac{\rho}{2} \left(\frac{Q_i(t)}{A_c} \right)^2 \\ &\quad + \zeta_{s_{\text{in}}} \frac{\rho}{2} \left(\frac{Q_i(t)}{A_{s_{\text{in}}}} \right)^2 + \frac{1}{C_s} \int_{-\infty}^{t} Q_i(\tau) - Q_o(\tau) d\tau \\ &= K_1 \cdot Q_i(t)^2 + \frac{1}{C_s} \int_{-\infty}^{t} Q_i(\tau) - Q_o(\tau) d\tau \end{aligned} \tag{2.16}$$

where K_1 is:

$$K_1 = \left(\frac{\zeta_d}{3A_d^2} + \frac{\zeta_c}{A_c^2} + \frac{\zeta_{s_{\text{in}}}}{A_{s_{\text{in}}}^2} \right) \cdot \frac{\rho}{2} = R_d + R_c + R_{s_{\text{in}}} \tag{2.17}$$

Since $Q_i(t)^2$ is a nonlinear term, a linearization is required in order to be able to perform a Laplace transform of the differential equations and set up a transfer function.

$Q_i(t)^2$ can be linearized using a first-order Taylor-series expansion about an operating point $(\overline{t}, \overline{Q}(t))$, or more shortly $(\overline{t}, \overline{Q})$. The first-order expansion is:

$$Q^2 \simeq \overline{Q}^2 + \left.\frac{d(Q^2)}{dQ}\right|_{Q=\overline{Q}} (Q - \overline{Q}) = \overline{Q}^2 + 2\overline{Q} \cdot \widehat{Q} \tag{2.18}$$

Inserting 2.18 in 2.16 yields:

$$\Delta p(t) = K_1 \left(\overline{Q}_i^2 + 2\overline{Q}_i \widehat{Q}_i(t)\right) + \frac{1}{C_s} \int_{-\infty}^{t} \widehat{Q}_i(\tau) - \widehat{Q}_o(\tau) d\tau \tag{2.19}$$

By differentiating both sides, equation 2.19 can be written as:

$$\frac{d\Delta p(t)}{dt} = 2K_1 \overline{Q}_i \frac{d\widehat{Q}_i(t)}{dt} + \frac{1}{C_s} \widehat{Q}_i(t) - \frac{1}{C_s} \widehat{Q}_o(t) \tag{2.20}$$

Using the Laplace transform on equation 2.20 gives:

$$s\Delta p(s) = \left(2K_1 \overline{Q}_i s + \frac{1}{C_s}\right) \widehat{Q}_i(s) - \frac{1}{C_s} \widehat{Q}_o(s) \tag{2.21}$$

Hereby the equation that describes loop A in the equivalent diagram of figure 2.2 is obtained. In order to obtain a similar equation for loop B, equation 2.3 is expanded:

$$\begin{aligned} \frac{1}{C_s} \int_{-\infty}^{t} Q_i(\tau) - Q_o(\tau) d\tau = {} & \zeta_{s_{\text{out}}} \frac{\rho}{2} \left(\frac{Q_o(t)}{A_{s_{\text{out}}}}\right)^2 + \frac{l_t \lambda_t}{D_{H_t}} \cdot \frac{\rho}{2} \left(\frac{Q_o(t)}{A_t}\right)^2 \\ & + \zeta_{90^\circ} \frac{\rho}{2} \left(\left(\frac{Q_o(t)}{A_{b3}}\right)^2 + \left(\frac{Q_o(t)}{A_{b4}}\right)^2\right) \\ & = K_2 \cdot Q_o(t)^2 \end{aligned} \tag{2.22}$$

where K_2 is:

$$K_2 = \left(\frac{\zeta_{s_{\text{out}}}}{A_{s_{\text{out}}}^2} + \frac{l_t \lambda_t}{D_{H_t} A_t^2} + \frac{\zeta_{90^\circ}}{A_{b3}^2 + A_{b4}^2}\right) \cdot \frac{\rho}{2} = R_{s_{\text{out}}} + R_t + R_{b3} + R_{b4} \tag{2.23}$$

Inserting equation 2.18 in equation 2.22 in gives:

$$\frac{1}{C_s} \int_{-\infty}^{t} \widehat{Q}_i(\tau) - \widehat{Q}_o(\tau) d\tau = K_2 \left(\overline{Q}_o^2 + 2\overline{Q}_o \widehat{Q}_o(t)\right) \tag{2.24}$$

and differentiating both sides, the equation can be written as:

$$\frac{1}{C_s}\widehat{Q}_i(t) - \frac{1}{C_s}\widehat{Q}_o(t) = 2K_2\overline{Q}_o\frac{d\widehat{Q}_o(t)}{dt} \tag{2.25}$$

Making the Laplace transform of equation 2.25 and isolating $\widehat{Q}_i(s)$ gives:

$$\widehat{Q}_i(s) = 2K_2C_s\overline{Q}_o s\widehat{Q}_o(s) + \widehat{Q}_o(s) \tag{2.26}$$

By combining equation 2.21 and 2.26 the transfer funtion from $\Delta p(s)$ to $\widehat{Q}_o(s)$ can be obtained:

$$\frac{\widehat{Q}_o(s)}{\Delta p(s)} = \frac{1}{2}\cdot\frac{1}{2K_1K_2C_s\overline{Q}_i\overline{Q}_o\cdot s + K_1\overline{Q}_i + K_2\overline{Q}_o} \tag{2.27}$$

2.1.1 Inserting Values in the Transfer Function

By means of the tunnel blueprints and the preceding formulas, the equivalent resistances of the different tunnel components have been computed and the results are listed in table 2.2.

Equivalent Resistance	Value [$\mathrm{N}\cdot\mathrm{s}^2/\mathrm{m}^6$]
R_d	0.99
R_c	0.24
$R_{s_{\mathrm{in}}}$	0.24
$R_{s_{\mathrm{out}}}$	0.12
R_t	0.19
R_b	27.28

Table 2.2: Equivalent resistances of the tunnel model.

Under the assumption that the temperature T is constant 293 K, γ is 1.1 and R_{air} is 287 $\frac{\mathrm{J}}{\mathrm{K}\cdot\mathrm{mol}}$, the capacitance of the stagnation box C_s is found to be $1.84\cdot10^{-4}\ \frac{\mathrm{mol}\cdot\mathrm{m}^3}{\mathrm{J}}$.

The operating point of $\overline{Q}_o$ has been chosen to be 15 $\frac{\mathrm{m}}{\mathrm{s}}$. To calculate the value of $\overline{Q}_o$ the velocity $v(t)$ must be multiplied by the area of the test section since:

$$v(t) = \frac{Q(t)}{A} \tag{2.28}$$

The area A of the test section is 0.186 m^2 which yields that $\overline{Q}_o = 2.80\frac{m^3}{s}$.

In steady state the volume flow is the same all through the tunnel and thus $\overline{Q}_i$ has been chosen to be the same as $\overline{Q}_o$.

Writing the transfer function in standard form and inserting the values gives:

$$\frac{Q_o(s)}{\Delta p(s)} = \frac{G}{\tau s + 1} = \frac{0.0061}{0.0014s + 1} = \frac{4.36}{s + 714.3} \tag{2.29}$$

which correctly implies that the gain G is less than one.

2.2 Motor, VLT and Fan

In general the system that produces the pressure difference $\Delta p(t)$ in the wind tunnel consists of the elements in figure 2.4.

Figure 2.4: A representation of the system from input voltage $u(t)$ to increase in air pressure $\Delta p(t)$.

The VLT is a control device for an induction motor. It basicly consists of an inverter, which is a power electronical device, and some control and protection circuitry. The inverter is capable of producing a 3-phase AC-voltage that vary in both angular frequency and amplitude, from a DC-voltage. This means that the VLT first rectifies the AC line voltage, and then produces the desired 3-phase AC voltage for the motor. The conversion ratio can be controlled either locally on the VLT or through various remote channels. One example is to use an analogue control voltage input $u(t)$ to specify the output angular frequency, which is used in the project [cdrom, Data sheet].

The purpose of using a VLT as an input source to the motor is to aid an easier use of the motor. This is for instance done through the use of current control, which means that the angular frequency is not applied or removed instantaneously, but gradually (controlled by the current drawn or induced by the motor), to minimize power loss and wear [cdrom, Data sheet]. This leads to the assumption that the dynamics of the motor is overshadowed by the dynamics dictated by the VLT. This means that setting up a complete dynamic model of the motor itself is inappropriate.

Further, since the fan is mounted on the motor shaft, and the motor and fan are assembled and placed in the tunnel, it is not possible to make any measurements on the motor itself. Consequently the model of the VLT, motor and fan is derived on the basis of measurements rather than first principles.

To arrive at the block diagram representation of the system in figure 2.5, the first thing that is noted is that the conversion from the control voltage $u(t)$ to

the angular frequency $\omega_{\text{motor}}(t)$ is linear (here the signal amplitude is discarded) [cdrom, Data sheet]. The conversion constant K_{VLT} is found from knowledge of the maximum input control voltage $u_{\max}$ and the maximum output angular frequency $\omega_{\text{VLT,max}}$, and the fact that the angular frequency is proportional with the control voltage. This gives the following:

$$K_{\text{VLT}} = \frac{\omega_{\text{VLT,max}}}{u_{\max}} \tag{2.30}$$

For the motor it is known that the angular frequency $\omega_{\text{motor}}(t)$ of the shaft has a linear dependence of the input frequency $\omega_{\text{VLT}}(t)$ in steady state (kilde: Per Sandholt). It is assumed that this can be generalized, when using a VLT. The proportionality constant K_{motor} is found to be:

$$K_{\text{motor}} = \frac{\omega_{\text{motor,max}}}{\omega_{\text{VLT,max}}} \tag{2.31}$$

As stated above, the dynamics of the motor and VLT is assumed to be dominated by the VLT dynamics. It is known that the combination of a motor and VLT can be modeled as at least a second order system (Kilde: Jakob Stoustrup). To arrive at a simple model the second order approximation is used.

$$\frac{\omega(s)}{u(s)} = \frac{K}{s^2 + 2\zeta\omega_n s + \omega_n^2} \tag{2.32}$$

In this formula the constant K is the product of the constants K_{VLT} and K_{motor}.

The last element of the model is the fan. Since it has not been possible to find any specific data on the fan, it is assumed that the fan is not equally efficient at all $\omega_{\text{motor}}(t)$. This implies that the fan has a characteristic dependent of $\omega_{\text{motor}}(t)$ in both a direct and an indirect way, like in equation 2.33.

$$\Delta p(t) = K_{\text{fan}} \cdot \omega_{\text{motor}}(t) \cdot f(\omega_{\text{motor}}(t)) \tag{2.33}$$

Here K_{fan} is a conversion constant from $\omega_{\text{motor}}(t)$ to $\Delta p(t)$ and $f(\omega_{\text{motor}}(t))$ is an unknown function of $\omega_{\text{motor}}(t)$. The expression in equation 2.33 is nonlinear and since none of the terms in equation 2.33 are known, the fan is modelled as a general nonlinear function.

From this it is now possible to write down the following block diagram:

The result is a model outline that has to be verified. Further, some of the terms in the model need to be calculated to complete the model. This is done through model estimation based on measurements.

2.2.1 Calculating Values in the Model

From the previous section, two values can be calculated: the constants in equation 2.30 and 2.31. From the available data on the VLT and motor where it is

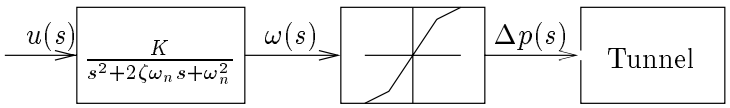


Figure 2.5: A block diagram representation of the model outline for the motor, VLT and fan.

given that $\omega_{\text{VLT,max}} = 2\pi \cdot 50$ and $u_{\max} = 10$, the following can be calculated:

$$\begin{aligned} K_{\text{VLT}} &= \frac{50 \cdot 2\pi}{10} = 31.42 \ \tfrac{\text{rad}}{\text{s}\cdot\text{V}} \\ \omega_{\text{motor,max}} &= 2935 \text{ rpm} \sim 307.35 \ \tfrac{\text{rad}}{\text{s}} \Rightarrow \\ K_{\text{motor}} &= \frac{307.35}{50 \cdot 2\pi} = 0.98 \end{aligned} \tag{2.34}$$

From this the constant K can be found to be:

$$K = K_{\text{VLT}} \cdot K_{\text{motor}} = 30.74 \ \tfrac{\text{rad}}{\text{s}\cdot\text{V}} \tag{2.35}$$

This can then be inserted in the transfer function in equation 2.32.

2.3 The Overall Model

Combining the anticipated transfer function of the fan and motor with the derived linearized transfer function of the wind tunnel gives an overall transfer function from VLT input to volume flow in the test section, illustrated on figure 2.6.

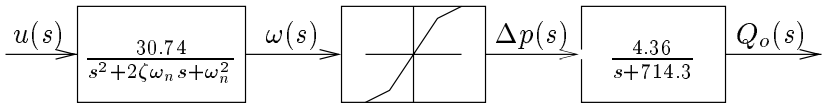


Figure 2.6: A block diagram representation of the overall model.

Where the transfer function from the VLT input to the pressure increase over the fan and motor is composed of two terms: the transfer function of the VLT and motor and the expression of the pressure increase due to the fan's angular velocity. The latter is a nonlinear function of ω and therefore not a real transfer function but for illustrative purposes it has been included all the same and will be dealt with later.

2.4 Model from Measurements

To obtain the actual model of the system, measurements of the real system have been taken. When the tests was made the pressure sensors where not

installed on the wind tunnel, so these measurements where made with a hand held instrument to measure the air velocity. More information about the measurement are given in appendix C.

The outcome of these measurements are loaded into MATLAB™ where the signals are filtered and the transfer function is estimated using the toolbox SENSTOOLS.

With these measurements the transfer functions from the VLT input voltage to the air velocity in the test section can be estimated. This is considered to be the whole system. Two step-up and one step-down measurements have been made to investigate possible differences. A computer with an I/O-card is used for the measurements.

The software used to carry out these measurements consists of two parts. One part for handling the output signal and one part to acquire the measured signal. The output signal from the I/O-card is a voltage step generator with 10 steps of one volt each. Every step lasts for 10 seconds. A MATLAB™ plot of the step generator for a step-up is shown in figure 2.7.

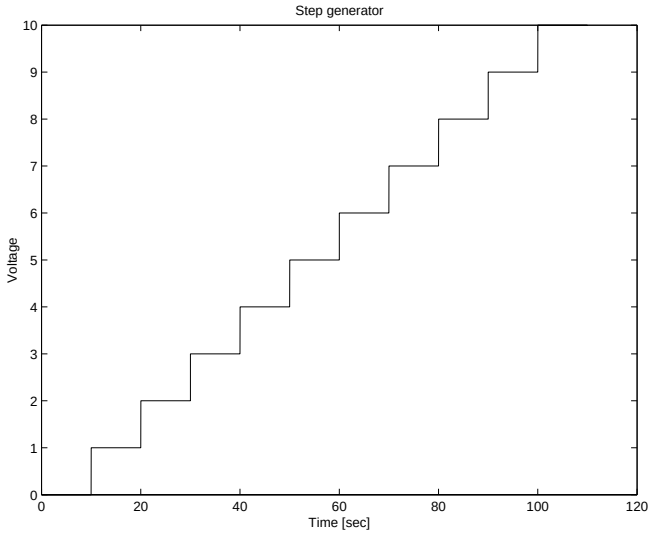


Figure 2.7: The signal from the step generator from the I/O-card.

The signal measurement handling in the software stores the measured signal in an ASCII-file, as also the input step is stored in an ASCII-file.

2.4.1 Results from the Step-up

The results from the step-up measurements are loaded into MATLAB™. A plot of one of the step-up measurements is shown in figure 2.8.

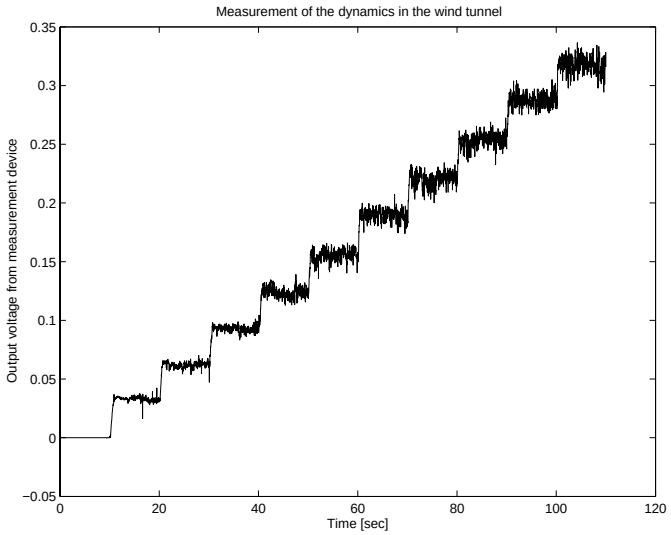


Figure 2.8: Plot of one of the step-up measurements.

This signal has to be filtered for SENSTOOLS to make a good fit. The filter used for these measurements is a second-order Butterworth lowpass filter (see appendix D).

SENSTOOLS is invoked with the following input arguments: The input step vector, the filtered output vector and a time vector. Then SENSTOOLS estimates the parametres of the transfer function. The filtered signal and the SENSTOOLS fit are shown in figure 2.9.

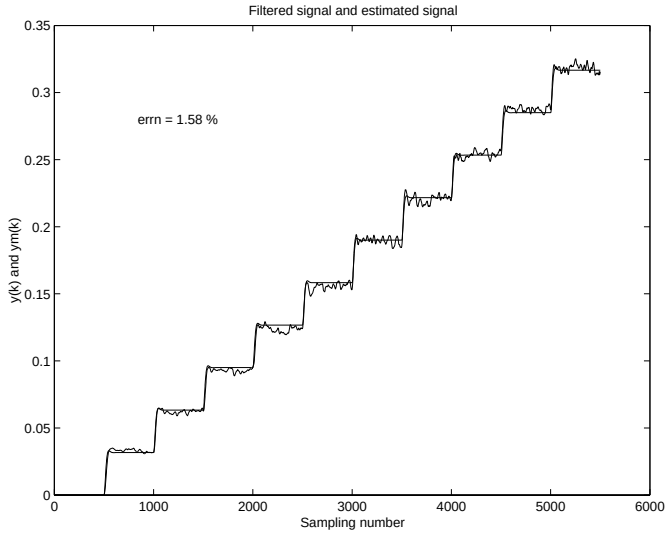


Figure 2.9: The filtered and estimated signal from a step-up measurement. The estimation error is written on the figure.

The estimation error is shown on the figure and it reads 1.58%. This error is quite acceptable, since the filtered signal contains some noise variations that Senstools is not supposed to make a fit to. Further, a certain degree of nonlinearity can be observed in the measurement, but Senstools can not estimate this nonlinearity and it is assumed to be so small that the system can be considered linear.

The parameters from the two step-up fits are averaged and the transfer function for a step-up is:

$$\frac{0.623}{s^2 + 6.44s + 19.8} \tag{2.36}$$

This test was made with an offset in the VLT. In the calculations this offset was removed, because the filter and Senstools must have a signal starting in 0. The offset resulted in the output frequency of the VLT being 5 Hz when the input signal was 0 V and 50 Hz when the input signal was 10 V. 5 Hz is 10 % of 50 Hz, and it affects only the gain of the transfer function with a factor 1.1, so the final transfer function for the step-up becomes:

$$\frac{0.685}{s^2 + 6.44s + 19.8} \tag{2.37}$$

Also a measurement with a large input step (from zero to maximum) was made to see if this transfer function also is usable for large steps on the motor. This measurement was made without the offset to look at the dynamics when the motor starts from zero. This step is plotted together with the step response to equation 2.37 multiplied by ten, and shown in figure 2.10.

The figure shows that the dynamics are slower when the steps become larger. It also shows that there is a short delay, or high order dynamics of the system, when the motor starts from zero. This is illustrated in figure 2.11 where two steps of one volt each are shown. The slowest response is from zero to one volt and the fastest is from four to five volt.

This delay or higher order dynamics must be accounted for in the control system.

2.4.2 Results from the Step-down

The step-down measurements have been made in exactly the same way as the step up measurements. One test was considered to be enough, since the two measurements from step-up where much alike. The same equipment and butterworth-filter where used. The filtered and the estimated signal are shown in figure 2.12.

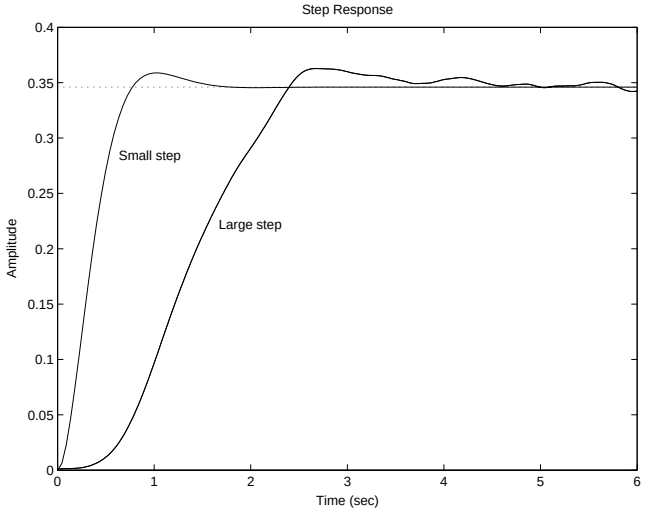


Figure 2.10: The step response from the measurements with small steps together with the step response from a large step.

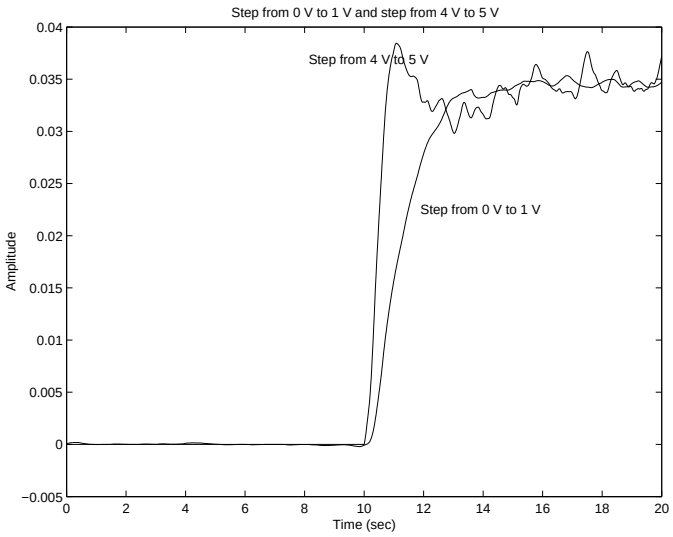


Figure 2.11: Two steps of one volt each. The slowest of them is a step from 0 V to 1 V and the other is from 4 V to 5 V.

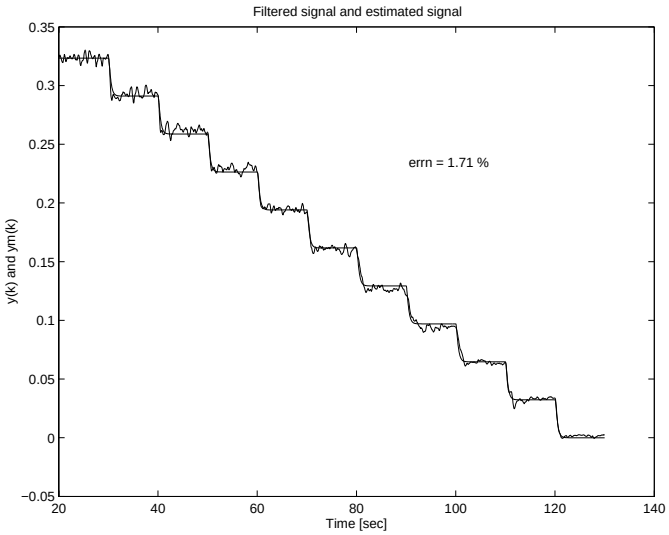


Figure 2.12: The filtered and estimated signal from the step-down measurement.

During this measurement the same offset as in the step-up measurement was set in the VLT. It means that the transfer function for the step down also is multiplied with 1.1. The final transfer function for the step-down measurement is shown in equation 2.38.

$$\frac{0.657}{s^2 + 9.31s + 18.5} \tag{2.38}$$

It is assumed that the step response is slower when the input step becomes larger in the step-down, as it was in the step-up. To compare the step-up and the step-down transfer functions, the step responses are plotted in figure 2.13.

Both systems are approximated to second-order systems, and it can be seen that the damping ratio is larger for the step-down model than for the step-up model. This, in turn, means that the dynamics of the step-up transfer function are faster than those of the step-down transfer function.

2.5 The final Model

The overall model fitted to the measurements is a second-order system, whereas the derived model yields a third-order system.

However, since the dynamics of the wind tunnel is assumed to be much faster than that of the VLT and motor (the time constant $\tau_{\text{tunnel}} = 6$ ms whereas the rise time of the whole model $t_{r,\text{whole model}} \simeq 400$ ms), the dynamics of the whole model are dominated by the VLT and motor model and therefore the tunnel model can be disregarded. This statement is further supported by the

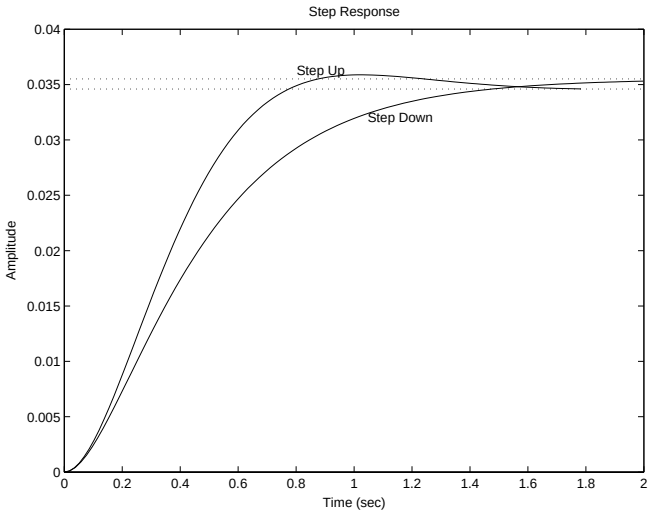


Figure 2.13: Step response for the step-up and step-down transfer functions.

following considerations.

The rise time t_r of a second-order system is given by:

$$t_r \simeq \frac{1.8}{\omega_n} \tag{2.39}$$

and the maximum rise time (the transition time from 10% to 90% of the final value) dictated by the VLT is 0.8 s (see figure 2.14). According to equation

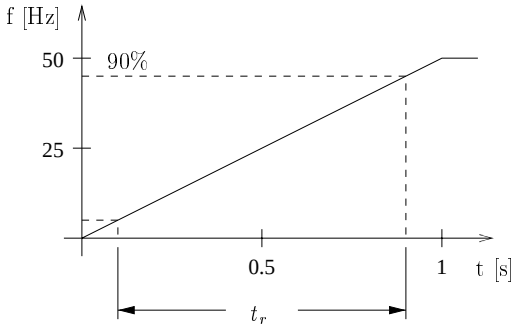


Figure 2.14: The rise time t_r dictated by the VLT.

2.39 the maximum natural frequency $\omega_{n,\max}$ is therefore 2.25 $\frac{\text{rad}}{\text{s}}$, limited by the VLT.

The poles of a standard second-order system, inserting the value of $\omega_{n,\max}$, are given by:

$$\frac{K}{s^2 + 2\zeta\omega_n s + \omega_n^2} = \frac{K}{s^2 + 4.5\zeta s + 5.0625} = \frac{K}{(s + p_1)(s + p_2)} \tag{2.40}$$

where:

$$p_1, p_2 = -2.25\zeta \pm 2.25\sqrt{\zeta^2 - 1} \tag{2.41}$$

For an underdamped systems the damping ratio ζ is real number between 0 and 1, so the poles vary between $p_1 = p_2 = -2.25$ and $p_1, p_2 = \pm 2.25j$ as shown in figure 2.15.

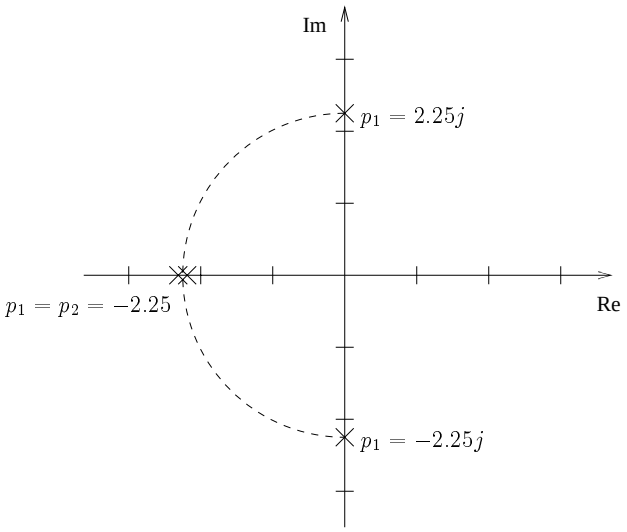


Figure 2.15: Poles of the second-order system when ζ vary between 0 and 1.

Using equation 2.41 and the fact that in systems where two poles are separated by more than a factor 10 in magnitude, the slowest becomes dominant and the faster can be discarded [haugen, p 193], the following is stated.

$\zeta \leq 5$ As the damping ratio exeeds 1, the two poles move along the real axis, one moving increasingly faster towards $-\infty$ and the other gradualy slower towards the origin. In this situation the two poles of the motor and VLT are closer than a factor 10 together and the pole of the wind tunnel is about 30 times faster. This means that the pole of the wind tunnel is discarded and this gives a total system of second order.

$\zeta > 5$ All the poles are separated individually by more than a factor 10 making the slowest pole dominant. Therefore both the pole of the wind tunnel and the faster of the two other poles can be omitted. This gives a first order system.

Looking at the measured step response, it can be seen that they have the following characteristics: Both responses have a horisontal tangent in 0 and the step-up reponse has a slight overshoot. This implies that the transfer function is a second order system corresponding to $\zeta < 5$. This means that the pole of the wind tunnel, placed in s=714.3 $\frac{\text{rad}}{\text{s}}$ can be descarded.

The proposed nonlinear element of the derived model is assumed to have influence on the gain of the response. The average gain in each step of the filtered curve in figure 2.9 is plotted against a straight line in figure 2.16.

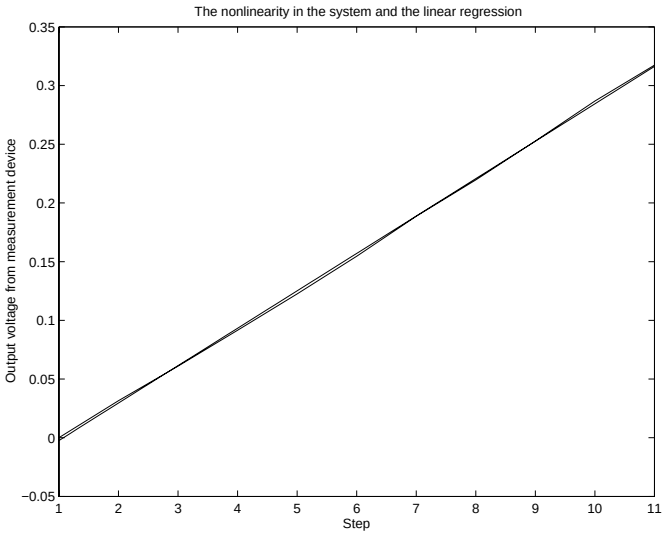


Figure 2.16: The average of each step in the step up caracteristic plotted a straight line.

From the figure it can be seen that the nonlinearety is insignificant and, thus, can be considered a constant K_{fan}.

2.6 Sensors

The wind tunnel is equipped with pressure transducers at the end of the test section to provide the controller with the appropriate feedback. Futher a temperature sensor is installed to inform the user about the temperature in the wind tunnel.

2.6.1 Pressure Transducers

The pressure transducers measure the dynamic pressure caused by the air velocity at the end of the test section and the air velocity is found from calculations based on the pressure.

The pressure transducer are differential analogue pressure transducers of type 1151 and 1351 from the Rosemount Alphaline Pressure Transducers Family. The analogue output range from 4 to 20 mA DC and is linear with the process pressure [cdrom, Data sheet].

The transducers are placed on the outside of the wind tunnel and the pressure is measured through a grid of pitot tube at the end of the test section.

The pitot principle is shown on figure 2.17 where the total pressure (p_t) is the sum of the static (p_s) and the dynamic (p_d) pressure.

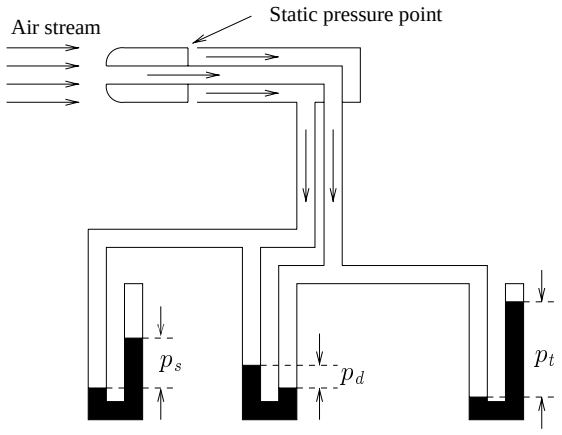


Figure 2.17: Figure of the pitot tube principle [ståbi, p 428].

p_s is the ambient pressure and p_d is the pressure caused by the air velosity.

The pressure transducers measure the difference between the total and static pressure and have the dynamic pressure as an output. The dynamic pressure is used in equation 2.42 to find the air velocity [ståbi, p 428].

$$p_d = \frac{\rho_{\text{air}}}{2} v^2 \Leftrightarrow v = \sqrt{\frac{2 \cdot p_d}{\rho_{\text{air}}}} \tag{2.42}$$

The pressure transducers use the principle sketched on figure 2.17 to find the dynamic pressure as the difference between the total and the static pressure.

How the pressure transducers works is illustrated on figure 2.18 where a draft of the cross section of the Rosemount δ-Cell$^{\text{TM}}$ sensor is shown.

A pitot tube is connected to either side of the δ-Cell$^{\text{TM}}$, transferring the static and the total pressure respectively into an oil filled chamber containing a sensing diaphragm. When a voltage difference is applied between the walls of the chamber and the sensing diaphragm, an electric capacitance will exist between these.

A difference in the static and the total air pressure will cause the sensing diaphragm to move towards the connection with the lowest air pressure. The capacitance will change proportionally with the displacement of the sensing diaphragm.

The capacitance is converted to an electric current, which is a linear function of the dynamic air pressure, and with this function, the magnitude of the air velocity can be obtained with the use of equation 2.42.

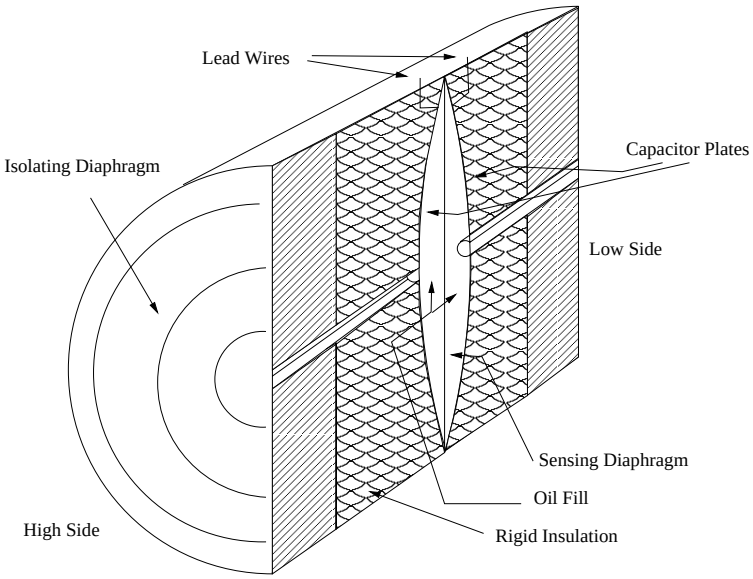


Figure 2.18: Cross section of a Rosemount δ-Cell™ [cdrom, Data sheet].

2.6.2 Temperature Sensor

The temperature sensor is a PT100 sensor, which is an international standard of positive temperature coefficient (PTC) platinum resistance thermometer sensors. It is a class A sensor with a tolerence of $\pm$ 0.1% and the output is approximately linear from 0° C to 100° C.

A PT100 sensor uses the principle that the resistance of platinum depends on the temperature. The measuring method is to send a constant current through the platinum wire and measure the voltage over it. By Ohm's law the resistance is found and by using the temperature characteristics of the platinum wire the temperature can be determined.

Chapter 3

Demand Specification

In this chapter the system is described, and the extends of the project will be delimited to the system this project is about. The internal and external interfaces are described in order to make a modulization.

3.1 System Description

The system contains the following major elements:

- PC.
- VLT (Motor interface).
- Motor.
- I/O card.
- Temperature sensor.
- Pressure transducers.

The interactions between the elements of the system are shown in figure 3.1.

PC: The PC is the main controller of the system. It carries out the following tasks:

- Implements the system's control algorithms.
- Controls the motor through the VLT.
- Draws a graphical user interface (GUI).
- Reads inputs from transducers connected to the I/O card.

VLT: The VLT is the motor driver controlled by the PC.

Motor: The motor is equipped with a fan which produces the circulation of air inside the tunnel. The angular velocity and thereby the air pressure is controlled by the VLT.

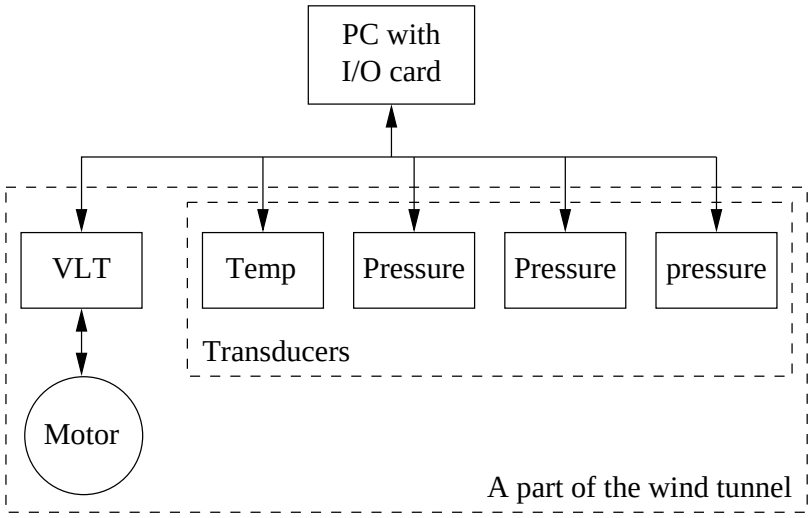


Figure 3.1: The construction of the main system. Only the parts dealt with in this project is included in this figure.

I/O card: The I/O card is the interface between the PC and the transducers and between the PC and the VLT.

Temperature sensor: The temperature sensor measures the temperature in the wind tunnel and convert it to an electrical signal to be acquired by the PC via the I/O card.

Pressure transducers: The pressure transducers measure the air pressure and convert it to electrical signals that can be captured by the I/O card.

3.2 System Function

The system should control the air velocity inside the wind tunnel. When the user provides the control system with a set of desired values of air velocity, the system must bring the tunnel in the desired state. The system should then inform the user when the desired setpoint values have been reached.

The user should have the opportunity to make the air velocity increase/decrease in a ramp function. It should be possible to provide the system with data about the inclination of the ramp.

All the measured data must be stored in a log file.

3.3 System Limitations

Control of the cooler connected to the wind tunnel will not be considered in this project.

The control system is only designed to operate correctly when the test object is small. If the test objects are too large the air velocity in the testing area will increase and the air velocity measurements will not match the actual air velocity in the test section. If large objects are tested the user must convert the measurements to an equivalent air velocity in the testing area.

It is presumed that there will not be made any changes on the wind tunnel, since this will affect the model of the system, and thereby the behaviour of the controller.

3.4 The Future of the System

The system will be implemented on a PC connected to the wind tunnel already built. The system will be the final product and will be in use for further use of the wind tunnel.

The provided software is open source, with no warranty and can be freely used, altered and distributed.

3.5 User Profile

The user is expected to be a member of the staff or technical personnel from the Institute of Energy Technology at Aalborg University. It is assumed that the user has a certain knowledge about fluid mechanics. The user does not need to know anything about control engineering.

The user will have to know how the wind tunnel operates and know about the air velocity inside the wind tunnel. Otherwise no special qualifications are required.

3.6 Demands to the Development Process

The control software and the Graphical User Interface (GUI) will be implemented on a PC with Windows® 2000 software using C. The modeling and simulations of the system and its components will be made in MATLAB™.

The communication between a PC and the VLT is carried out using an analogus output on the I/O card.

The final product, including the documentation, must be handed in on the 31. May 2002.

3.7 Parts to be Delivered to the Customer

Below is listed the parts to be delivered to the customer:

- System documentation (this report).
- A CDROM containing the application software.
- A user manual in English.

3.8 Assumptions

The following items are assumed to be available trough the developing process:

- PC with Windows® 2000 as the operating system and all necessary software.
- VLT - Danfoss VLT 6000 HVAC.
- Motor - Leroy-sommers model LS160MP with a Novenco model ACN 630 air fan.
- I/O card - NuDAQ model PCI 9112, 12 bit + device drivers and libraries for C.
- Pressure transducers - Rosemount model 1351 (6 units).
- Temperature sensor, type PT100.
- Wind tunnel.
- Consultans from the Institute of Energy Technology.

3.9 Specific Demands

The areas of dynamic for air velocity goes from 0 $\frac{\text{m}}{\text{s}}$ to 30 $\frac{\text{m}}{\text{s}}$. The accuracy of the controller must be $\pm$ 1%.

PC: The PC must be able to communicate with the I/O card via a PCI slot. It must be able to maintain the GUI, and therefore contain the software needed to draw the GUI.

The log file should contain the data in an ASCII .csv file. It contains the measurement data obtained at any specific time. The measurement data will be mean values of the air velocity and temperature at the end of the test section.

Controller: The controller must ensure that the overshoot of the air velocity in the wind tunnel must be within $\pm 1\%$ and a rise time at least 15% slower than the VLT ramp. The sample rate is chosen to 50 Hz.

VLT: The VLT must be able to receive input from the PC and control the motor.

Motor: The motor must be able to accelerate/decelerate the air with reference to the dynamic demands.

I/O: 7 analogue inputs are required, one for each transducer. The pressure transducers are arranged in pairs where one transducer measures the velocity from 0 $\frac{\text{m}}{\text{s}}$ to 12 $\frac{\text{m}}{\text{s}}$ and one measures from 0 $\frac{\text{m}}{\text{s}}$ to 30 $\frac{\text{m}}{\text{s}}$.

The control system has to select the right transducer at the present velocity.

Further one analogue output is required for the VLT control. The I/O card must be accessed using function calls in C.

Pressure transducer: The pressure transducers must give an output voltage between 0 and 10 V DC each.

Temperature sensor: The temperature sensor must give an output voltage between 0 and 10 V DC.

3.10 Internal Interfaces

PC–I/O: A device driver for Windows® 2000 must be provided to make the I/O card accesseble using C functions. Transfer rate and datatypes is specified by the PCI bus standard.

I/O–VLT: The interface between the I/O card and the VLT is given by an analogue signal with voltage level 0–10 V DC.

I/O–Pressure transducer: The voltage level is 0–10 V DC.

I/O–Temperature sensor: The voltage level is 0–10 V DC.

3.11 External Interfaces

The external interface is the GUI, and it must contain the following:

- Text in English.
- The possibility to navigate through the interface using both mouse or keyboard.

- A readout of the current air velocity (in $\frac{\text{m}}{\text{s}}$) and air temperature (in $°C$) in the wind tunnel.
- An indication of when the air velocity has reached the desired value.
- A possibility for a user to type in a desired air velocity (in $\frac{\text{m}}{\text{s}}$) to the system.
- A possibility to make the air velocity in the tunnel follow a user defined ramp carateristic.
- A possibility to store a .csv file with logged information about the air velocity and temperature in the tunnel.

A suggestion for the graphical user interface is shown in figure 3.2.

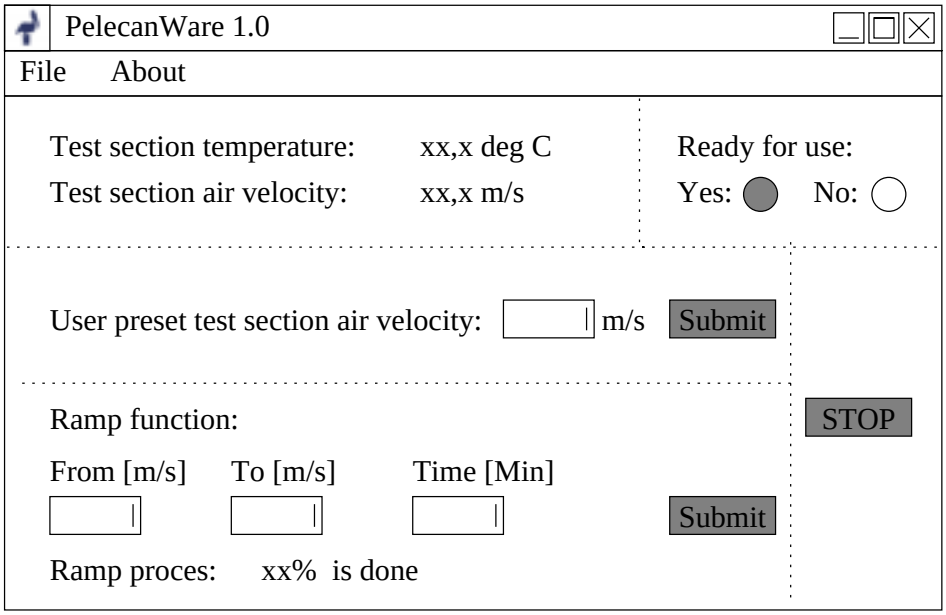


Figure 3.2: The user interface of the system.

3.12 Accept Test Specification

In the following the demands for the system are listed in two categories. One for the graphical user interface (GUI) and one for the control algorithm. After the list of demands, a specification of how the demands must be tested will follow.

Demands for the GUI

Demand 1: All text in the GUI must be in English.

Demand 2: All buttons and text fields must be accessible with both mouse and keyboard and functional.

Demand 3: The numbers typed in the text fields must be of the format xx.x, where the x'es are the numbers from 0 to 9 and the "." is a decimal delimiter. The numbers after the "." are optional. The range of input air velocity is from 0 to 30 $\frac{\text{m}}{\text{s}}$. If an invalid input is given, the GUI must display an error message.

Demand 4: The GUI must give the user an indication on whether the wind tunnel is "ready" or "not ready" for measurements. This must be done by displaying a red or green indicator respectively.

Demand 5: The software must produce an ASCII-text file in .csv format, containing logged information about the air velocity and temperature from the last invocation of the "Submit" button. The log file will be stored on the harddisk of the PC. A MATLAB™ program `PelecanPlot` can be used to load the data from the .csv file and to make a plot of the measurements.

Demands for the Control Algorithm

Demand 6: The sample frequency must ensure that the shortest rise/fall time is sampled at least 10 times.

Demand 7: The controller must not be so fast that the VLT ramp limits the speed. The accuracy of the controller must be $\pm 1\%$, that is a maximum of 0.3 $\frac{\text{m}}{\text{s}}$ at 30 $\frac{\text{m}}{\text{s}}$. A 0.3 $\frac{\text{m}}{\text{s}}$ step, corresponding to a 100 mV input step, takes 10 ms when the ramp time from 0 V to 10 V is set to 1 s. It means that the rise time and fall time of the controller must be at least $t_r > 0.08$ s as shown in figure 3.3.

Demand 8: The overshoot must be within the demand of $\pm 1\%$ accuracy.

Demand 9: The system must have no tracking error for ramp input.

Demand 10: The system must deliver a stable air velocity, within $\pm 1\%$ of the desired value, in the wind tunnel within the limits of 0–30 $\frac{\text{m}}{\text{s}}$.

Test Specification

Test 1: Start up the GUI and navigate through all the menus, buttons and pop up windows, verifying that they are in English.

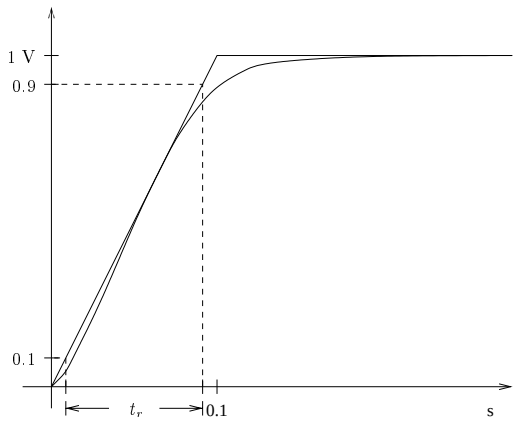


Figure 3.3: Rise time requirements for the system. Same requirements are used for the fall time. A VLT step of 1 V takes 0.1 s.

Test 2: Start up the GUI and navigate through the screen with only the mouse. Ensure that all areas can be reached. Repeat the test only using the keyboard. Ensure that all areas can be reached.

Test 3: Start the GUI. Move to a text field and type a legal number and see that the software accepts the value. Type a value that is out of range and see that the system gives an error and rejects the input. Type in strings that contain illegal symbols or are of an invalid format, and see that the system displays an error message and rejects the input.

Test 4: Start the GUI and type in a desired air velocity in the appropriate text field. Submit the value and verify that the air velocity in the wind tunnel starts building up. As the desired air velocity is submitted the indication "not ready" will be shown and when the air velocity reaches the desired value the indication will switch to "ready".

Test 5: Start the GUI and submit a legal air velocity in the text field. Let the air velocity build up and press the stop button after an appropriate time. Go to the file menu and choose the save option. Enter a file name and a location for the file and press "save". Open a MATLAB™ command window and choose the "import" option in the file menu. Find the newly saved file from the wind tunnel control program and ensure that it has the ".csv" extension. Load it into MATLAB™ with `PelecanPlot` and see that the output from `PelecanPlot` contains information of the time span, air velocity and temperature of the just performed test in three separate vectors.

Test 6: Use the procedure of test 5 to save and load a file into MATLAB™ and use the `plot` command to plot the air velocity response vs. the time vector and compute the sample time. Read the rise time from the plot and compute the number of samples on a rise time by multiplying with

the sample frequency. According to the demand this has to be at least 10.

Test 7 and 8: Use the procedure in test 5 to produce, save and load a positive step response into MATLAB™. Plot the response vs. time with the `plot` command. Verify that the overshoot present is within 0.1% of the steady state air velocity. Compare the submitted value with the last recorded values of the test, and see that they are the same within ±1%. Compute the rise time and verify that it fulfills the demand. Repeat the procedure for a negative step.

Test 9: Specify a ramp input to the system and submit the input. Use the procedure of test 5 to save and load the result into MATLAB™. View the resulting output with the `plot` command and compare it with the submitted ramp. The system response must follow the specified ramp within ±1%.

Test 10: Submit different values of air velocity for the system and let the same velocity be present for 3 min. Examine the resulting velocity with MATLAB™ and with a separate hand held instrument to measure the air velocity in the test section. The air velocity must not vary more than the specified ±1%.

Chapter 4

Modulization

The project is divided into six modules. Each module represents a main application of the system and can be designed and tested separately. The modules are specified below.

Module 1 - Setpoint Changer

The primary purpose of this module is to ensure that the motor does not have to respond to big steps. because the motor can be damaged by such steps.

When the system is started from zero it appears to act as a higher order system and introduces a sort of delay from the start signal is given until the sensors measure a pressure difference. The algorithm in this module also deals with this delay. Not taking this in to account will reduce the system performance as the controller will react very brutally to the delay and introduce a large overshoot and a long settling time.

Module 2 - Classical Controller

This module contains the design, simulation and implementation of two classical controllers: A PID controller and a PID controller together with an input filter design.

Module 3 - State-space Controller

This module describes an alternative to the classical controller in module 2. It deals with the design, simulation and implementation of a state space controller.

Module 4 - Choice of Controller

In this module the controllers from module 2 and 3 are compared and the best is chosen to be implemented in the system.

Module 5 - Hardware Configuration

Module 5 deals with the practical configuration of hardware to handle the sensors and actuators in the wind tunnel.

Module 6 - Software

Module 6 describes the implementation of the chosen controller on the PC. It also describes the design of the GUI and the other underlying software.

Chapter 5

Module 1 - Setpoint Changer

From the analysis of the motor, fan and VLT it is evident that the system reacts differently when the setpoint is changed from zero than from any other setpoint. In figure 2.11 the step response of the system starting from rest is compared to the step response of the system initially in motion. From this it can be seen that the system shows signs of higher order dynamics when started from rest and that it reacts slower. It has been observed that the system reacts slower if a big leap in setpoint is made compared to a small change in setpoint. This can be seen from figure 2.10. This yields the need for a module to control the change in setpoints.

5.1 Module Considerations

From the preceding paragraph it can be seen that the setpoint changer module must ensure a slow start of the system. This can be done in several ways, but the controller of the system must be taken into account. If the controller contains an integral term, the controller has to follow the startup process. If not, it will produce a substantial overshoot when it is put into operation since no error has been accumulated in the integrator.

5.1.1 Module Demands

The demand specification and the previous section yield the following demands to the setpoint changer module:

- The setpoint transition from 0–10 V or from 10–0 V must be slower than 10 s.
- The current limiter of the VLT must not come into action under setpoint change.

The demand to the transition time from 0–10 V is based on experimental observations, to ensure that the VLT current limeter is not put into action.

5.2 Module Design

The module is chosen to be a setpoint changer, placed before the system as in figure 5.1. This position ensures that an integral term in the controller of the system always gets input from the sensors, such that an overshoot caused by an integral term in the controller is avoided. Since this module controls the setpoint, it is independent of the controller placed in the system, which means that the system controller can be made as fast as the demands allow.

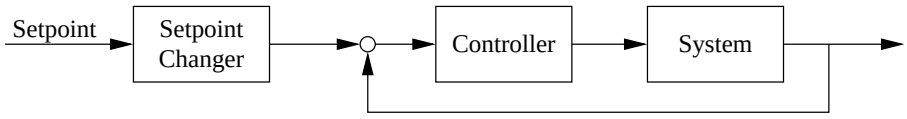


Figure 5.1: The placement of the setpoint changer module in the total system.

To control the setpoint, the setpoint changer must contain the following parts:

Decision function: The setpoint changer must be able to decide if the newly submitted setpoint is larger or smaller than the previous submitted setpoint.

Ramp function: This function decides the rate of change in setpoint. The module must have an upward and downward going ramp for each of the two cases in the decision function.

Stop function: This function stops the changing of the setpoint when the new setpoint value is reached.

The parts are connected as shown in figure 5.2.

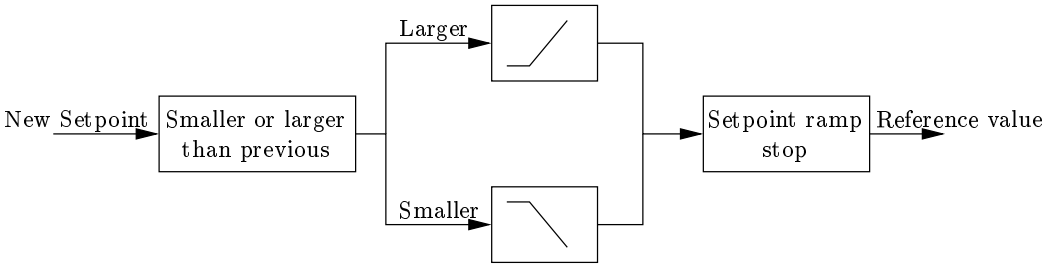


Figure 5.2: The principle of the setpoint changer module.

Transforming this description of the module into a flow chart, yields the structure of figure 5.3. It is programmed in a separate C function, where the variables `SetPoint_New`, `SetPoint_Old` and `Reference` are the newly and previously

submitted setpoint and the reference input to the control algorithm, respectivly. When the stop button is pressed, the setpoint value 0 will be passed to the module.

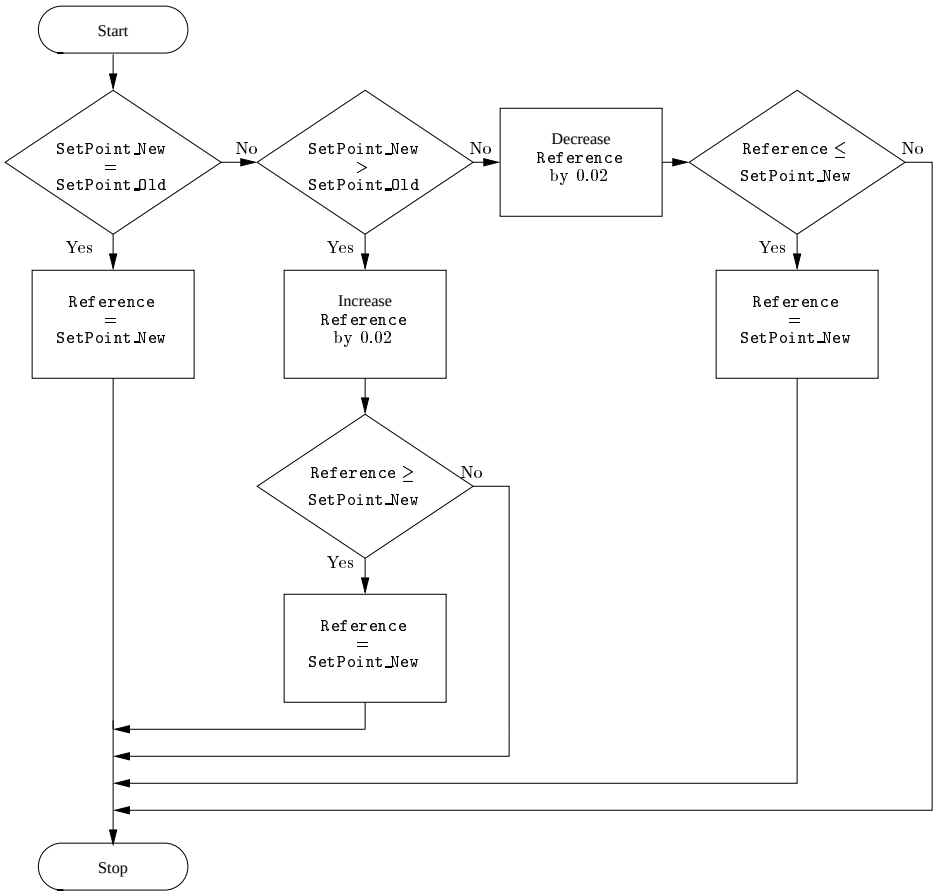


Figure 5.3: A flow chart representation of the start setpoint changer module.

The execution rate of the routine follows the sample frequency (f_s) since the communication with the VLT follows the same rate. Therefore the function itself does not contain any timer functions. This means that the function needs to test if a new setpoint is submitted. This is done by the first of the decision blocks in the flow chart. If the setpoint is changed the steps in which it is updated is given as:

$$\begin{aligned} \text{Number of time steps} &= f_s \cdot 10 = 500 \text{ Steps} \\ \text{Number of voltage steps} &= f_s \cdot 10 = 500 \text{ Steps} \\ \text{Voltage resolution} &= \frac{\text{Max Voltage}}{\text{Number of voltage steps}} = 0.02 \ \frac{\text{V}}{\text{Step}} \end{aligned} \tag{5.1}$$

Since the setpoint is altered in discrete steps of 0.02 V it is obvious that this module will produce a quantization error. This error must be kept small in order to keep the transition as smooth as possible. Further this quantization

implies that it is not certain that the accurate setpoint can be obtained. Therefore the module must update the setpoint until it is reached or just past it. As this happens, the reference variable is set to the submitted setpoint value to ensure that the reference equals the setpoint.

5.3 Test of Setpoint Changer

The tests of the setpoint changer module are carried out with the use of a test program that includes the module. The test program contains the module and is able to read out the resulting reference value to the screen. The test contains the following parts:

Part 1: A setpoint greater or smaller than the previous must result in an upward or downward going ramp respectivly.

Part 2: The inclination of the ramp must result in a reference rise time of at least 10 s

Part 3: The reference value must be kept stable after it reaches the setpoint.

The first part of the test is performed by passing different setpoint values to the module and observe the direction of the ramp. In the second part of the test the values of `Reference` and `SetPoint_Old` are set to 0 and `SetPoint_New` to 10. The transition time is found by counting the number of times the reference is updated before it reaches 10. This number is multiplied by the sampling period and compared to 10 s. The third part of the test is carried out by submitting a setpoint value. As the setpoint is reached the output reference must not be altered.

Performing the tests showed that the module is able to make a ramp in the right direction and stop as it reaches the submitted setpoint. Further, the transition time from 0 – 10 s is calculated. The number of steps is found to be 500 Steps, and with a sample time of 50 Hz the transition time yields:

$$\text{Ramp time} = \frac{1}{f_s} \cdot 500 = 10 \text{ s} \tag{5.2}$$

5.4 Conclusion

The test of the module showed that it was able to perform the desired tasks. The transition time was not measured directly, but since it is presumed that the sample rate is fixed, this procedure is valid.

Chapter 6

Module 2 - Classical Controller

In this chapter two different controller design strategies will be considered. The first is a pure classical controller and the other is a classical controller in conjunction with an input filter. The two controller designs must perform the same task, which is to make the output of the plant follow the input reference. The two strategies are shown in figure 6.1.

In the first part of the chapter the controller without input filter is designed. Through this section the different design ideas are described in detail. The design of the controller with filter follows the same structure. The chapter is rounded off with a part describing the simulation and test of the two controllers in order to compare the two controller strategies.

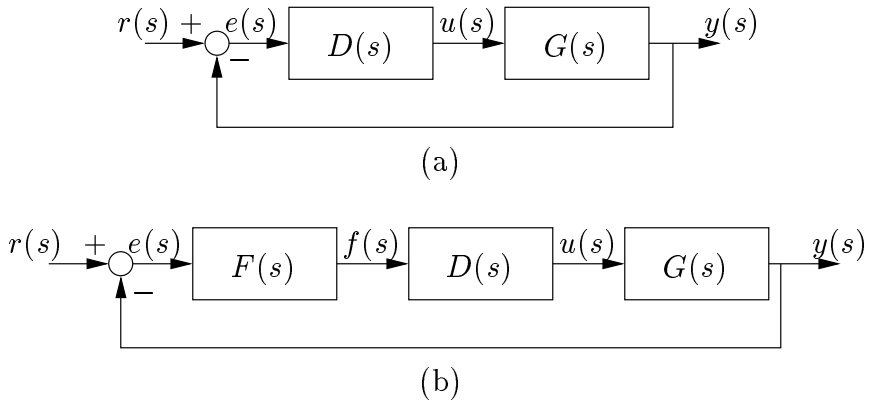


Figure 6.1: The two closed-loop controllers to be designed in this chapter. (a) is without filter and (b) is with the filter in the feedforward path.

6.1 Module Demands

From the demand specification the following demands to the module can be set:

- Rise time $t_r \approx 0.2$ s.
- Resulting closed-loop bandwidth $\omega_b < 15 \ \frac{\text{rad}}{\text{s}}$.
- Allowable overshoot $M_p = \pm 1\%$.
- Stable air velocity within $\pm 1\%$ of the desired value.
- The module must be made with a classical controller.

The rise time demand is a result of the chosen sample rate. Since the maximum sample rate is specified in the demand specification and the number of samples in one rise time is specified, the minimal rise time can be calculated. The resulting bandwidth is specified to make it possible to realize the controller in a computer. A large bandwidth results in a demand for a high sample rate. Since the sample rate is limited in the demand specification a maximal bandwidth is specified.

6.2 Module Considerations

The controller to be designed in this chapter is a PID controller. The need for the integral term is based on the steady state error demand. The derivative term is needed to make the controller react fast enough without overshoot. This means that the controller can be written as:

$$D(S) = K\left(T_d s + \frac{1}{T_i s} + 1\right) = K\left(\frac{T_d T_i s^2 + T_i s + 1}{T_i s}\right) \tag{6.1}$$

The design procedure for this module is based on the use of root loci together with pole/zero maps and step responses. The root loci will be used to argument for a specific pole/zero placement and to give a hint about a desired gain. Since the poles cannot be directly placed with the use of a classical controller, the gain will have to be adjusted to yield the final design. This is done with the use of root loci together with step responses where the rise time and overshoot demand will be examined. In the end the bandwidth is found to ensure that the demand is followed. All this is at first done with a continuous-time prototype controller. This controller is transformed into a difference equation for implementation on the PC.

To design the two controllers, the step-up transfer function ($G_u(s)$) is used. This is possible since the step-up and step-down transfer functions are similar in structure in terms of poles and zeros. $G_u(s)$ is chosen since it is faster than the step-down transfer function ($G_d(s)$) which is most critical in terms of rise time demands. When the designs are completed it must be verified that it is usable for the step-down transfer function as well.

6.3 Design of Controller without Filter

The system has the transfer function shown in equation 2.37 which is repeated here for convenience:

$$G_u(s) = \frac{0.685}{s^2 + 6.44s + 19.8} \tag{6.2}$$

From this it can be seen that the system introduces a pair of complex poles in $p_1, p_2 = -3.2 \pm 3.07j$ and no zeros. It can be seen from equation 6.1 that the PID controller contains a real pole p_3 placed in the origin and two zeros, z_1, z_2. The two zeros of the controller can be placed freely in the complex plane to give, together with the gain, the desired dynamic response of the closed-loop system.

To determine an appropriate placement of the zeros the following knowledge of root loci is used: If the number of real poles and zeros to the right of a selected testpoint is odd the locus will be located on the real axis in the testpoint. Further, the fact that poles always move towards either a zero or $\pm\infty$ as the gain increases is used. From this it is obvious that the zeros must be placed in the left half plane to prevent the poles from becoming unstable. This leaves the following possible placements of the zeros:

Case 1: As a complex conjugate pair. A special case is to place the zeros to cancel out the complex poles p_1 and p_2 of the system.

Case 2: As real and distinct zeros.

The special case of case 1 means that the influence of the complex poles of the system is removed from the dynamic response. This gives a system dominated by the real pole from the controller that will move towards $-\infty$ along the real axis. This means that the system will act as a first order system. To be able to exactly cancel out p_1 and p_2 it is obvious that the model must be very accurate at all times. Therefore this zero placement is considered impractical in this context.

Placing the zeros as a complex pair in general means that the first order behaviour will depend on the gain. If the gain is selected to be of such size that p_3 is moved more than a factor 10 away from the complex poles, the oscillary behaviour of the complex poles will be dominant. This means that a controller that fulfills the rise time demand will produce an overshoot.

The result of placing the zeros at two distinct points on the real axis yields the pole/zero map in figure 6.2.

From the knowledge of root loci it can be derived that the root locus will be located on the real axis to the left of z_2. The locus will not be on the real axis between z_2 and z_1 and then again on the real axis from z_1 to the origin. This means that the poles can all be moved to the real axis to give a system

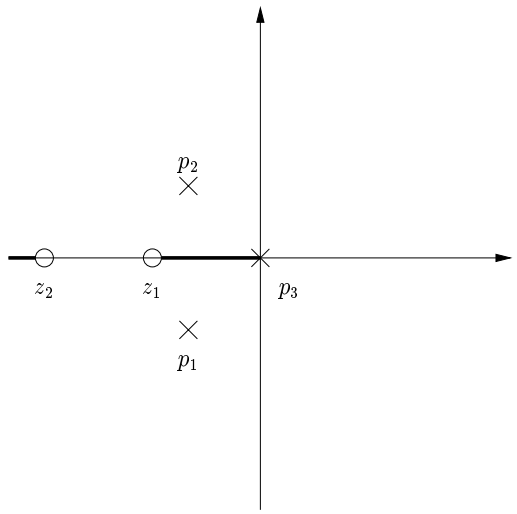


Figure 6.2: The pole/zero map of case 2.

without overshoot. The resulting root locus is shown in figure 6.3, where it is obvious that a relatively large gain is required to make the poles real. If the

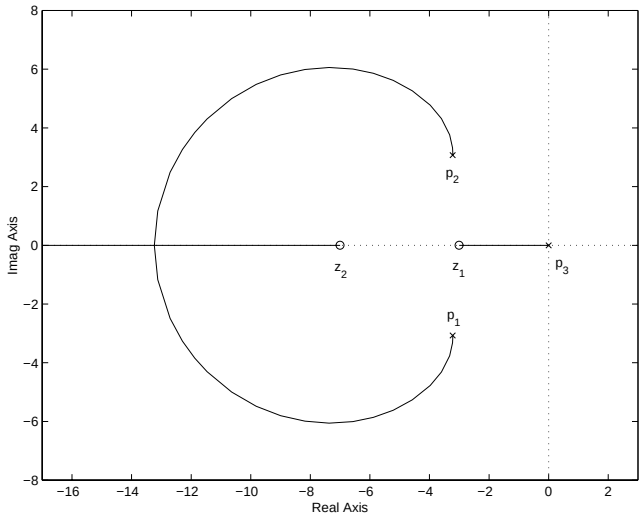


Figure 6.3: The root locus of case 2.

amount of gain required to move the poles is too high, a real zero can be used to damp the oscillation of the complex poles. The size of the gain is bounded by the bandwidth requirement, since larger gain yields a larger bandwidth.

After considering the different placements of z_1 and z_2 it is chosen to use case 2. This is done because it is not very likely to result in an unstable system.

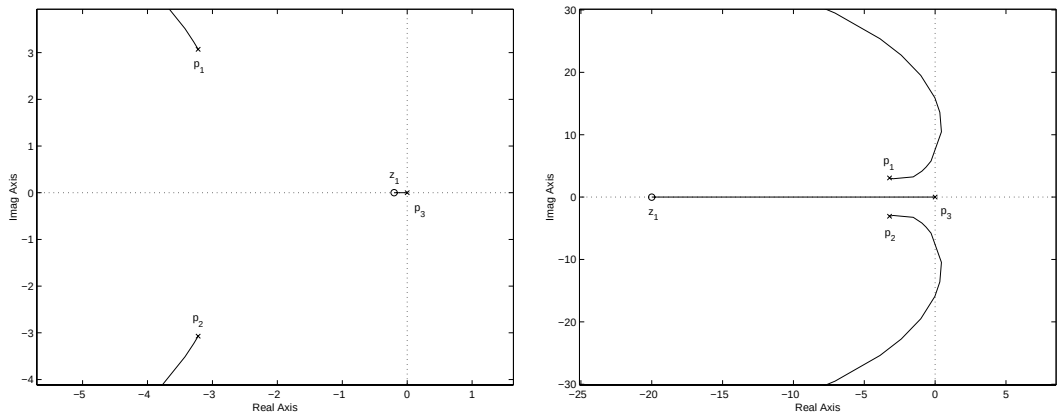
To make the zeros real, the numerator of $D(s)$ in equation 6.1 must have a

positive discriminant which means:

$$T_i^2 - 4 \cdot T_i \cdot T_d \cdot 1 > 0 \quad \Leftrightarrow \quad T_i > 4T_d \tag{6.3}$$

After pointing out the guidelines for the pole and zero placement the iterative process of finding the exact placement and the resulting gain can begin. This is done using the MATLAB^{TM} commands `rlocus` and `rlocfind`.

At first the placement of the slowest zero z_1 is considered.



(a) (b)

Figure 6.4: The root locus of the closed-loop transfer function $T(s)$ with z_1 placed either close to (a) or away from (b) the imaginary axis. z_2 is chosen to be $10 + z_1$ in both situations.

In part (a) of figure 6.4 it can be seen that the complex poles of $T(s)$ move into the right half plane at certain gains. If the gain is selected low enough to prevent this, the dynamics of the complex poles will dominate the response and yield a response with possible overshoot. From part (b) it can be seen that the stability is ensured. Instead p_1 and p_2 can not be moved very far without the use of high gains meaning that the first order dynamics of the real pole will be the dominant and that the step response will be slow. As a compromise it is chosen to place z_1 near the real part of p_1 and p_2.

The location of z_2 decides how far p_1 and p_2 have to "travel" to reach the real axis. The further z_2 is away from z_1 the more gain is needed to make p_1 and p_2 real. But if the two poles are close together the integral and derivative term of the PID will not be working optimal. To compromise between these two cases it is chosen that $T_i \approx 5 \cdot T_d$.

From this it is chosen that $z_1 = -3$ and $z_2 = -7$. To calculate T_i and T_d from these zeros the coefficients of the numerator in equation 6.1 are matched with $(s - z_1)(s - z_2)$.

$$
\begin{aligned}
s^2 - (z_1 + z_2)s + z_1 z_2 &= K(T_i T_d s^2 + T_i s + 1) \Leftrightarrow \\
T_i = \frac{-(z_1 + z_2)}{K} \qquad T_d &= \frac{1}{-(z_1 + z_2)} \\
K &= z_1 z_2
\end{aligned} \tag{6.4}
$$

This results in $T_i = 0.476$ and $T_d = 0.1$.

The gain is adjusted to give the desired result, by using the command `rlocfind` and the step response. The first thing noted is that the poles cannot all be real without using a large gain which is unacceptable due to the bandwidth requirements. Instead it is chosen to let the poles become complex and then control the overshoot with z_1 and the gain. To find the proper gain, different values are tested and the resulting step response of the closed-loop transfer function $T(s)$ is examined in terms of rise time and overshoot. The resulting step response is shown in figure 6.5.

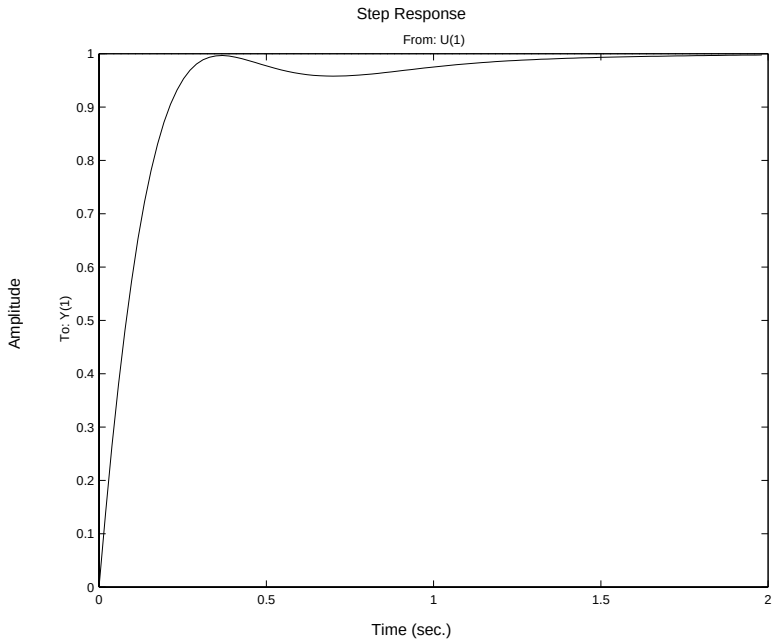


Figure 6.5: The resulting step resonse of the PID controller design.

It shows that the controller has no overshoot and that the rise time fulfills the demand. Finally the bandwidth of $T(s)$ is examined using the MATLAB^{TM} command `ltiview`. It is found that the bandwidth is $\omega_b = 10.4 \ \frac{\text{rad}}{\text{s}}$ which is inside the limits.

The resulting gain is 108 which means that the final continuous-time PID controller has the following form.

$$
D(s) = 108 \left(\frac{0.0476s^2 + 0.4762s + 1}{0.4762s} \right) \tag{6.5}
$$

It results in the pole/zero map in figure 6.6 where the real part of the complex poles is placed near z_2. Further, the slowest pole p_3 is moved towards z_1 to make the total response fast.

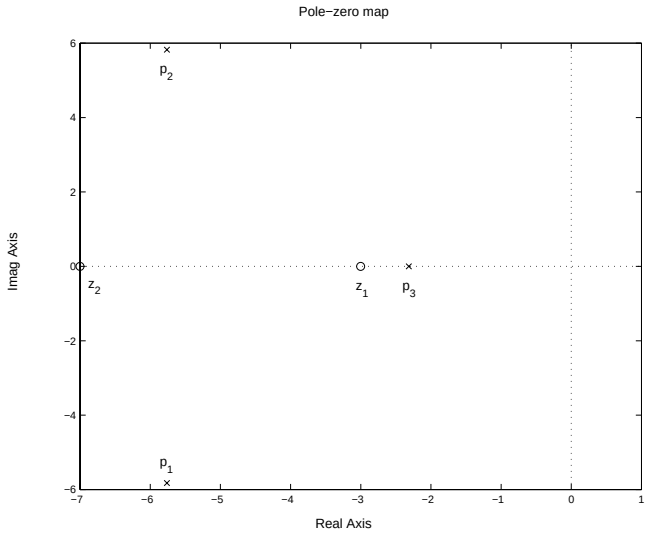


Figure 6.6: The resulting pole and zero location of the closed-loop transfer function.

To verify that the resulting continuous-time PID controller is usable together with $G_d(s)$ the step response and bode plot of the closed-loop transfer function is used. The results are presented in table 6.1 together with the results for $G_u(s)$.

System	t_r [s]	t_s [s]	M_p [%]	ω_b $[\frac{\text{rad}}{\text{s}}]$	e_{ss} step	e_{ss} ramp
$G_u(s)$	0.2	1.3	0	10.4	0	0.13
$G_d(s)$	0.3	0.5	0	6.71	0	0.12

Table 6.1: The different describing values of the continuous-time prototype controller. The calculation of the e_{ss} to a ramp input is based on a unity ramp input.

The table shows that the controller fulfills the demands with the step down transfer function as well.

6.3.1 Discretisation

The continuous-time controller transfer function $D(s)$ must be transformed into a discrete-time difference equation before it can be implemented on the PC. To make this transformation, the zero order hold (ZOH) method is used.

It is defined as:

$$H(z) = ZOH(H(s)) = (1 - z^{-1})\mathcal{Z}\left\{\frac{H(s)}{s}\right\} \tag{6.6}$$

where the $\mathcal{Z}$ denotes the z-transform, $H(s)$ is the continuous-time transfer function and $H(z)$ is the discrete-time transfer function. Applying this to the PID controller transfer function $D(s)$ yields the following, where T_s is the sample period:

$$\begin{aligned} ZOH(D(s)) = D(z) &= (1 - z^{-1})\mathcal{Z}\left\{\frac{D(s)}{s}\right\} \Leftrightarrow \\ D(z) &= (1 - z^{-1})\mathcal{Z}\left\{\left(\frac{K}{s} + \frac{K}{T_i s^2} + KT_d\right)\right\} \Leftrightarrow \\ D(z) &= K + \frac{KT_s z^{-1}}{T_i(1 - z^{-1})} + (1 - z^{-1})KT_d \end{aligned} \tag{6.7}$$

To compare the step response of $D(z)$ with the step response of $D(s)$, the values of T_d, T_i, T_s and K are inserted in $D(z)$. This is done to see if any additional overshoot is introduced by the discretisation process. In order to produce the step response of $D(z)$ the discrete-time equivalente of $G_u(s)$ is calculated through ZOH and the closed-loop transfer function $T(z)$ is calculated.

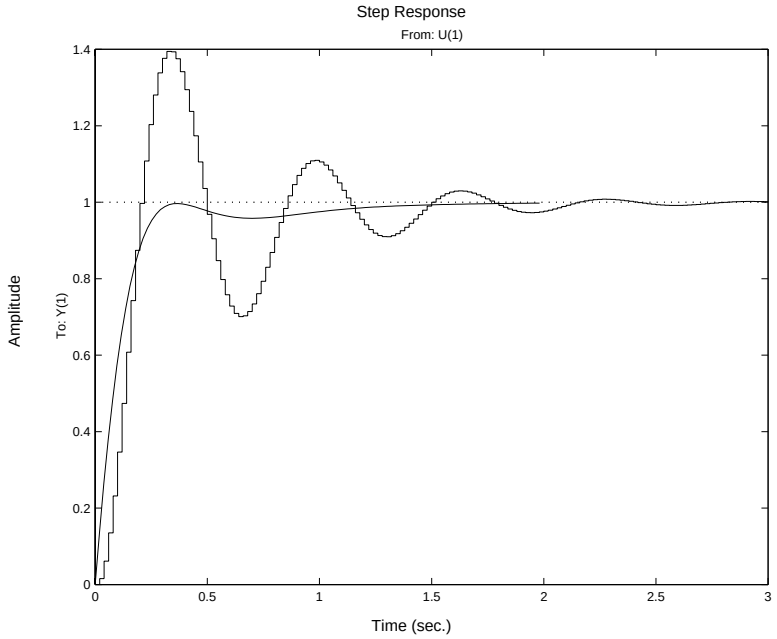


Figure 6.7: The plot of the step response of $T(z)$ together with the step response of $T(s)$.

In figure 6.7 it can be seen that the discretisation results in additional overshoot. This means that the discrete-time realisation of the controller must be

adjusted to fulfill the demands. It is clear from figure 6.7 that the gain needs to be reduced. The result of the adjustment is $K = 28$, $T_d = 0.083$ and $T_i = 0.375$. The reason for the movement of the zeros is to accomplish a better damping of the oscillations. The resulting continuous-time controller compared with the discrete-time realisation is shown in figure 6.8

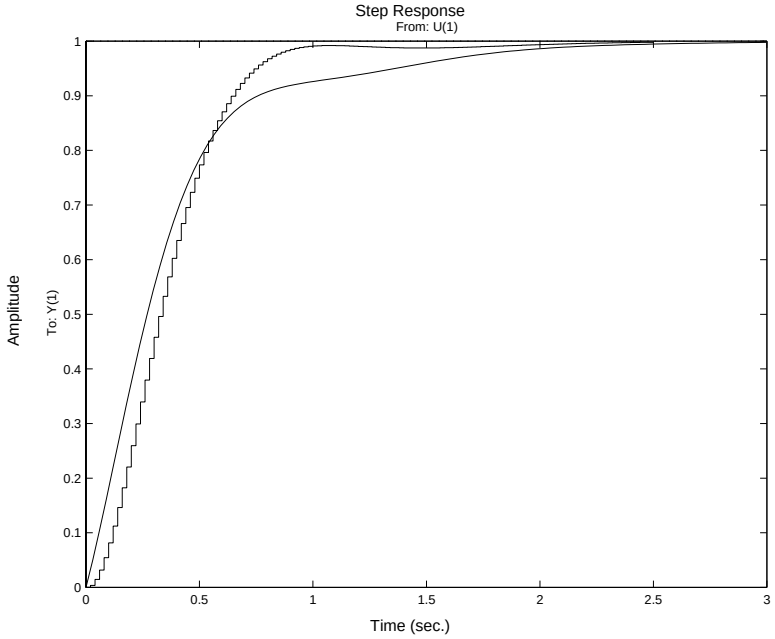


Figure 6.8: The step response of the adjusted discrete-time and continuous-time controller.

With these changes to the continuous-time controller the discrete-time realisation has the data shown i table 6.2.

System	t_r [s]	t_s [s]	M_p [%]	ω_b $[\frac{\text{rad}}{\text{s}}]$	e_{ss} step	e_{ss} ramp
$G_u(s)$	0.7	2.1	0	3.2	0	0.39
$G_d(s)$	0.7	1.9	1	2.7	0	0.38
$G_u(z)$	0.4	2.6	0	5.7	0	0.013
$G_d(z)$	0.5	1.8	0	4.4	0	0.013

Table 6.2: The different describing values of the continuous-time and discrete-time controller. The e_{ss} to ramp input is calculated for a unity ramp for the continuous-time prototype, and simulated in SIMULINK™ with the setpoint changer ramp for the discrete-time controller.

The table shows that the resulting discrete-time and continuous-time controllers are slowed down compared to controllers in table 6.1 to be able to

fulfill the demands after the discretisation. The final step is to transform the result into a difference equation.

Since $D(s)$ describes the transfer function from the error $e(s)$ to the control signal $u(s)$ the $D(z)$ can be rewritten to a difference equation:

$$\begin{aligned} D(z) = &\frac{u(z)}{e(z)} = K + \frac{KT_s z^{-1}}{T_i(1-z^{-1})} + (1-z^{-1})KT_d \Leftrightarrow \\ u(z)(1-z^{-1}) = & \\ & e(z)(K+KT_d) + e(z)\left(K\left(\frac{T_s}{T_i} - 1 - 2T_d\right)\right)z^{-1} \\ & + e(z)KT_s z^{-2} \end{aligned} \tag{6.8}$$

Applying the inverse z-transform ($\mathcal{Z}^{-1}$) to equation 6.8, the following difference equation is obtained:

$$\begin{aligned} u[n] = & \\ & e[n](K+KT_d) + e[n-1]\left(K\left(\frac{T_s}{T_i} - 1 - 2T_d\right)\right) \\ & + e[n-2]KT_d + u[n-1] \end{aligned} \tag{6.9}$$

By inserting the values for K, T_i and T_d as found in section 6.3, the final difference equation becomes:

$$u[n] = 30.33e[n] - 31.17e[n-1] + 2.33e[n-2] + u[n-1] \tag{6.10}$$

6.4 Design of Controller with Filter

The controller filter used in this context will, as stated earlier, be placed in the feedforward path of the control loop. This means that the purpose of the filter is to remove any noise on the error signal in the feedback controller. The largest noise sources in the system is considered to be the motor and VLT or more precisely the switch harmonics of the VLT and the AC-source voltage to the motor. The switching noise of the VLT is of very high frequencies, but the source voltage to the motor vary from 0–50 Hz. This means that the filter needs a very low cutoff frequency. On the other hand, the filter should have as little influence on the resulting closed-loop step response as possible, meaning that the poles of the filter must not be near 0 $\frac{\text{rad}}{\text{s}}$. As a compromise it is chosen that the filter $F(s)$ should be a second order lowpass filter with two real poles in $s = -5$. To make the filter have 0 dB gain in the pass band, the filter gets the following transfer function:

$$F(s) = \frac{25}{s^2 + 10s + 25} \tag{6.11}$$

A higher order filter will give better damping of the higher frequencies, but it will also introduce more poles. It is chosen as a compromise between high damping and number of poles to make a second order filter.

With this choice of filter the number of poles adds up to five, with the two complex poles $p_1, p_2 = -3.2 \pm 3.07j$ of the $G_u(s)$ transfer function, the controller pole $p_3 = 0$ and the filter poles $p_4, p_5 = -5$. The placement of the zeros is based on the same considerations as in the previous section, and again the choice is to make the zeros real. The final placement of the zeros is found to be $z_1 = -2$ and $z_2 = -7$. This results in the pole/zero map of figure 6.9.

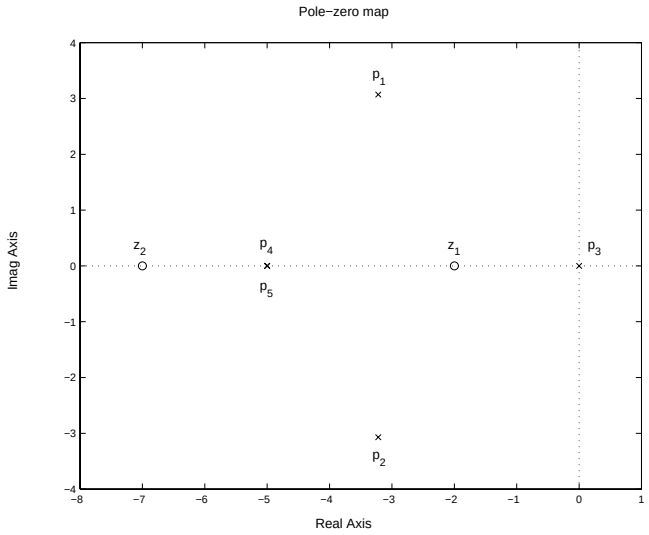


Figure 6.9: Pole/zero placement of the feedforward path of the controller using an input filter.

From the knowledge of root loci it can be seen that the root locus will be placed to the left of z_2 and to the right of z_1. Plotting the root locus shows that the complex poles of the system do not become real with time, but rather moves towards the right half plane. This means that the amount of gain used in the controller must be kept small enough to prevent p_1 and p_2 from becoming unstable. On the other hand, the gain must be high enough to move p_3 away from the imaginary axis, to make the closed-loop step response fast. The resulting root locus is shown in figure 6.10.

The gain is found to be 23 which gives the desired outcome. Then the transfer function for the continuous-time PID controller $D_f(s)$ can be calculated with the use of equation 6.4 to yield:

$$D_f(s) = 23 \left(\frac{0.0714s^2 + 0.642s + 1}{0.642s} \right) \tag{6.12}$$

The resulting closed-loop step response of the continuous-time system with

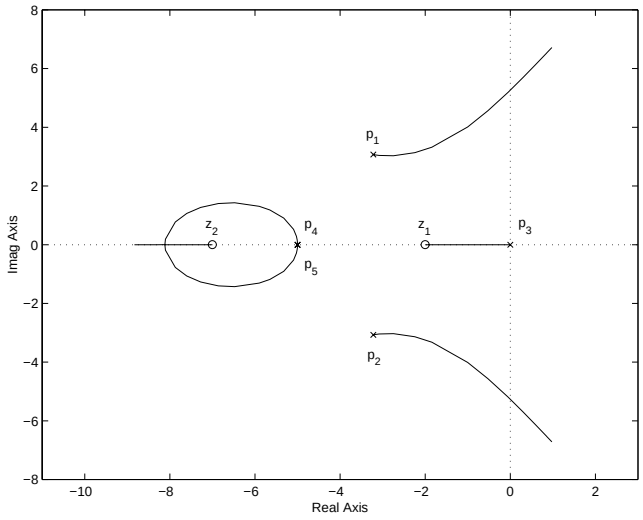


Figure 6.10: The root locus of the controller with input filter.

filter is shown in figure 6.11. It is possible to increase the gain even more in order to get the same response as for $G_u(s)$, but then the response to $G_d(s)$ results in an overshoot. The figure shows that the oscillations of the complex poles are not fully damped by the zero.

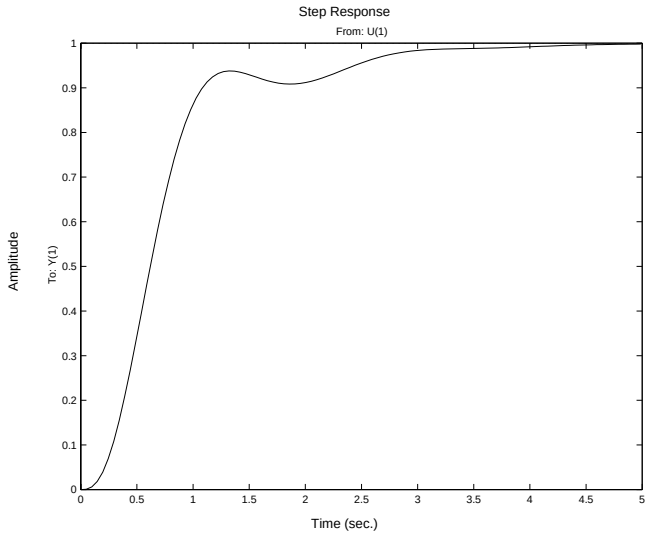


Figure 6.11: The step response of the controller with filter and the $G_u(s)$.

To verify this, the resulting controller and filter is tested together with $G_d(s)$ to see if it fulfills the demands. The results are presented in table 6.3 together with the results for $G_u(s)$.

From the table it is obvious that the combination of filter and controller results

System	t_r [s]	t_s [s]	M_p [%]	ω_b $[\frac{\text{rad}}{\text{s}}]$	e_{ss} step	e_{ss} ramp
$G_u(s)$	0.9	3.9	0	15	0	0.81
$G_d(s)$	1.0	2.9	0	11	0	0.78

Table 6.3: The describing values of the continuous-time controller with input filter. The e_{ss} for ramp input is calculated for a unit ramp.

in a very slow response that has large steady state errors for a ramp input.

6.4.1 Discretisation

The discretisation uses the same method as with the controller in the previous section. Here the ZOH is applied to both the filter $F(s)$ and the controller $D_f(s)$.

Applying the ZOH method to the filter gives the following:

$$\begin{aligned}
ZOH(F(s)) = F(z) &= (1-z^{-1})\mathcal{Z}\left\{\frac{F(s)}{s}\right\} \Leftrightarrow \\
F(z) &= (1-z^{-1})\mathcal{Z}\left\{\frac{25}{s\left(s^2+10s+25\right)}\right\} \Leftrightarrow \\
F(z) &= (1-z^{-1})\frac{z\left(z\left(1-e^{-5T_s}-5T_se^{-5T_s}\right)+e^{-10T_s}-e^{-5T_s}+5T_se^{-5T_s}\right)}{(z-1)\left(z-e^{-5T_s}\right)^2} \Leftrightarrow \\
F(z) &= \frac{z\left(1-e^{-5T_s}-5T_se^{-5T_s}\right)+e^{-10T_s}-e^{-5T_s}+5T_se^{-5T_s}}{\left(z-e^{-5T_s}\right)^2}
\end{aligned} \tag{6.13}$$

Reorganizing the equation and applying the inverse z-transform $(\mathcal{Z}^{-1})$ to equation 6.13 the following difference equation is obtained:

$$\begin{aligned}
f[n] = &\\
&e[n-2]\left(e^{-10T_s}-e^{-5T_s}+5T_se^{-5T_s}\right)+e[n-1]\left(1-e^{-5T_s}-5T_se^{-5T_s}\right) \\
&-f[n-2]e^{-10T_s}+f[n-1]2e^{-5T_s}
\end{aligned} \tag{6.14}$$

In equation 6.14, the $e[n]$ denotes the input signal to the filter and $f[n]$ is the filtered output. Inserting the value for the sample time $T_s = 0.02$ s results in the difference equation:

$$\begin{aligned}
f[n] = &\ 4.679\cdot10^{-3}e[n-1]+4.377\cdot10^{-3}e[n-2] \\
&+1.819f[n-1]+0.8187f[n-2]
\end{aligned} \tag{6.15}$$

The discrete-time controller has the form of equation 6.9, and with the values of K, T_i and T_d it becomes:

$$u[n] = 48.6e[n]-27.4e[n-1]+2.6e[n-2]+u[n-1] \tag{6.16}$$

The step response of the resulting system is shown in figure 6.12.

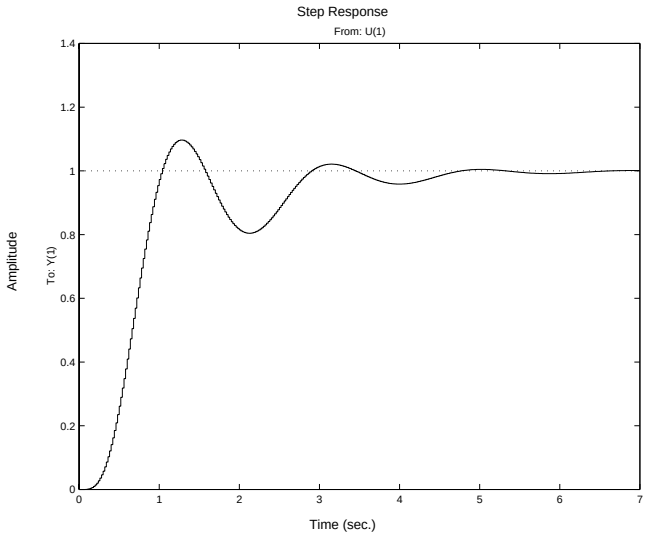


Figure 6.12: The step response of the discret-time closed-loop system with input filter $T_f(s)$.

Again some adjustments are needed to make the discrete-time controller fulfill the requirements. In this context the demands can be fulfilled by reducing the gain from $K = 23$ to $K = 19.5$. This gives the following step response (figure 6.13), difference equation (equation 6.17) and characteristics (table 6.4):

$$u[n] = 39.5e[n] - 23.2e[n-1] + 2.2e[n-2] + u[n-1] \tag{6.17}$$

System	t_r [s]	t_s [s]	M_p [%]	ω_b $[\frac{\text{rad}}{\text{s}}]$	e_{ss} step	e_{ss} ramp
$G_u(s)$	1.9	4.4	0	9.7	0	0.95
$G_d(s)$	1.3	3.4	0	11	0	0.93
$G_u(z)$	0.7	4.9	0	3.5	0	0.033
$G_d(z)$	0.8	4.0	0.5	2.8	0	0.33

Table 6.4: The different describing values of the continuous-time and discrete-time controller with input filter.

6.5 Simulation of the Classical Controllers

To verify the controller designs further a set of simulations are conducted. The simulations are produced with the MATLAB™ toolbox SIMULINK™ and are

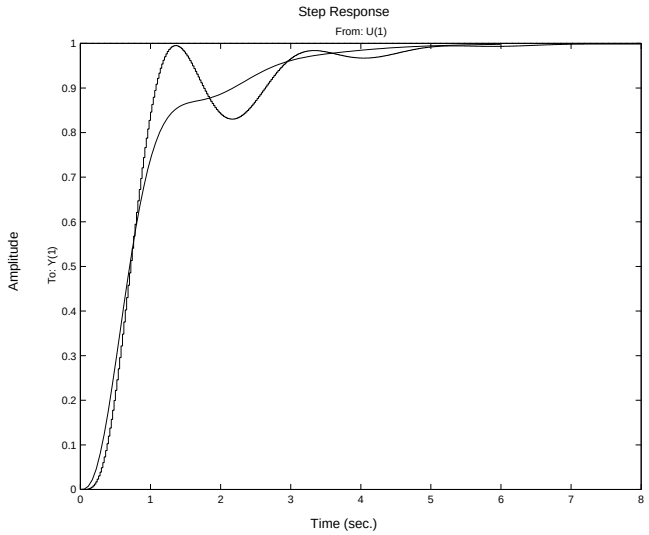


Figure 6.13: The step response for the continuous-time and discrete-time adjusted controller.

described together with the simulation model in appendix A.1. The simulations have to show the following:

1. The controllers' use of control signal.
2. The noise rejection ability of the controllers.
3. The response to ramp input.

The simulation model is shown in figure 6.14.

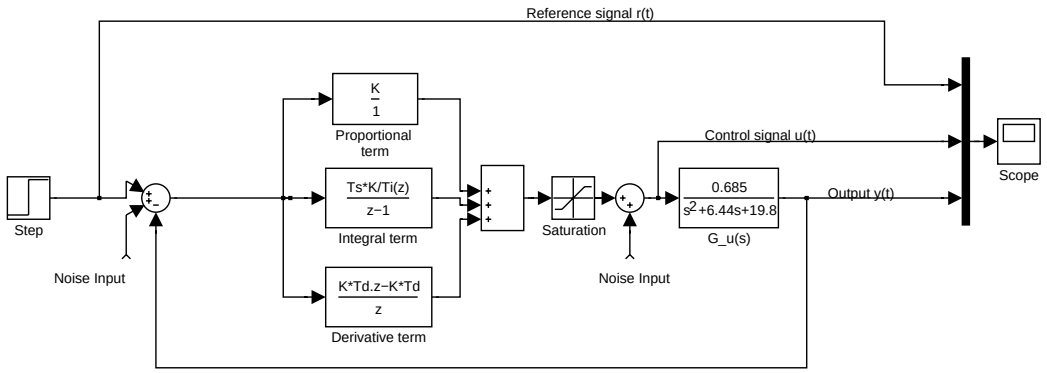


Figure 6.14: The simulation model used in all the simulations. To simulate the D_f controller the filter is added in the feedforward path.

6.5.1 Results

All plots from the simulations are shown in appendix A.1, and the results are summarized here.

Simulations for Controller Without Filter

The control signal has an approximate overshoot of 20% followed by the same amount of undershoot. The response to ramp input is a steady state error of 0.013 on the setpoint changer ramp, and 0.0013 on the slower ramp.

The results of the noise simulations are placed in table 6.5.

Sine	Feedback	Control Signal
5 Hz	Visible*	None
50 Hz	None	None
1 kHz	None	None*
Random	Dominant*	None*

Table 6.5: The effects of noise on the output signal for the design without filter. An (*) indicates that the control signal is varying dramaticly.

Simulations for Controller With Filter

The control signal has no overshoot. The response to a ramp is a steady state error of 0.033 for the setpoint changer ramp and 0.0033 on the slower ramp.

The results of the noise simulations are placed in table 6.6.

Sine	Feedback	Control Signal
5 Hz	None	None*
50 Hz	None	None
1 kHz	None	None
Random	Dominant	None*

Table 6.6: The effects of noise on the output signal for the design with filter. An (*) indicates that the control signal is varying dramaticly.

6.6 Test of the Classical Controllers

The tests of the two classical controllers has been performed by implementing the difference equations 6.10 and 6.17 in the control software that is described

in chapter 10 on page 77. The tests are further described in appendix F.

The tests show that the tunnel does not behave as it was expected from the model derived in chapter 2.4 on page 12. The primary reason for this deviation in the meassured signals is the calibration of the sensors. The sensor output from the six sensors are shown in figure F.1. Another observation done in the test was that the VLT went into current saturation much earlier than expected. This means the control signal could not be altered as fast as expected. Because the sensors have been mounted very late in the project period it has not been possible to take these situations into account in other ways than to try to implement the designed controllers and tune them until a satisfactory result was obtained.

The controller without filter had to be altered to work acceptably. It resulted in a $t_r = 1.8$ s and an overshoot af approximately 20%. The tests showed that the controller did not perform well on steps, but was acceptable with ramp inputs. The controller with filter did not need alterations to work acceptably. The results of the tests gave a slow controller with a $t_r = 4$ s, but no overshoot. It performes well on step inputs but is not as good as the controller without filter on ramp inputs. Both controllers do not settle on a steady setpoint but this can be due to noise on the measurements.

6.7 Conclusion

Through this chapter two classical controllers have been designed for the wind tunnel. The first is a standard PID controller and the other is a PID controller with input filter. It has been seen that it is possible to design the two controllers and the filter, by first making a continuous-time prototype and then discretisize it to implement it in the computer. It was found that directly implementing the discrete-time version of the continuous-time prototype did not give a satisfactory result. The reason for this is the use of the zero order hold method for discretisation. A better result could be obtained by the use of a different method, such as a bilinear transformation or by applying a discrete design method. The resulting controllers were then simulated with the MATLAB™ toolbox SIMULINK™ to give a satisfactory result.

Tests of the controllers showed that only the controller with filter could be implemented directly in the tunnel though it did not fully behave as expected. It showed problems with ramp input which might be caused by the filter. The problem could be solved by applying a discrete-time filter design. The controller without filter had to be altered in order to work. In both cases problems with the calibration of the transducers and the VLT current limiter is to blame.

Chapter 7

Module 3 - State-space Controller

State-space is a way of describing a system's input-output behaviour in terms of its internal states and it is based on the differential equations of the system. Moreover, state-space controller design has several important advantages over classical control methods:

- It can deal with multi-input, multi-output (MIMO) systems in the same way that it deals with single-input, single-output (SISO) systems.
- Even if some of the states of the process/plant cannot be measured, as is the case with the wind tunnel, the introduction of an observer (estimator) makes it possible to estimate the states with a good precision.
- A state-space controller can be designed using a divide and conquer approach, that is, it is possible to design the control law and the observer in separate steps.
- All the poles of the system can be placed freely.
- It uses linear algebraic expressions in the form of matrix gains which are very convenient to implement in computers.

7.1 General State-Space Description

The input-output relationship of a system can be represented with a state-space description as shown in figure 7.1 where the state-space equations are given by:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \end{aligned} \tag{7.1}$$

and the bold letters denote vectors or matrices.

Equation 7.1 is the general state-space description of a MIMO system. Since the wind tunnel has only on input and one output, all systems described in the following will be SISO systems.

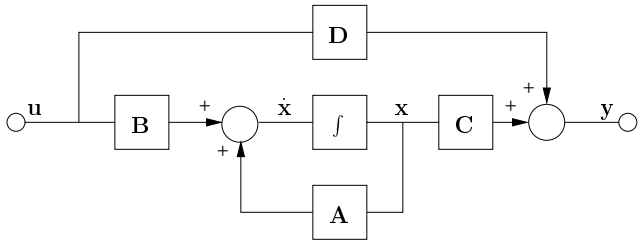


Figure 7.1: Block diagram of a general state-space description of a system.

7.1.1 Controllability and Observability

A system is said to be controllable if there is a coupling between the control signal u and each of the states of the system.

A formalized method to test for controllability is to compute the controllability matrix $\mathcal{C}$:

$$\mathcal{C} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \mathbf{A}^2\mathbf{B} & \dots & \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix} \tag{7.2}$$

If it has full rank, the system is controllable.

Observability refers to the ability to measure all states of the system. If the observability matrix $\mathcal{O}$ of a system has full rank, then it is said to be observable. $\mathcal{O}$ is given by:

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{n-1} \end{bmatrix} \tag{7.3}$$

7.2 State-Space Description of the Wind Tunnel

In section 2.4 on page 12 the step-up transfer function of the wind tunnel was found to be:

$$\frac{Y(s)}{U(s)} = \frac{0.685}{s^2 + 6.44s + 19.8} \tag{7.4}$$

from which the diffential equation of the system is found to be:

$$\begin{aligned} 0.685u &= \ddot{y} + 6.44\dot{y} + 19.8y \quad \Leftrightarrow \\ u &= \frac{1}{0.685}\left(\ddot{y} + 6.44\dot{y} + 19.8y\right) \end{aligned} \tag{7.5}$$

Then the two states x_1 and x_2 are chosen from the differential equation:

$$x_1 = \frac{1}{0.685}\dot{y}, \quad x_2 = \frac{1}{0.685}y \tag{7.6}$$

and the state derivatives become:

$$\begin{aligned} \dot{x_1} &= \frac{1}{0.685}\ddot{y} = u - 6.44x_1 - 19.8x_2 \\ \dot{x_2} &= \frac{1}{0.685}\dot{y} = x_1 \end{aligned} \tag{7.7}$$

from which the full state-space description of the wind tunnel can be derived:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} -6.44 & -19.8 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u \tag{7.8}$$

$$y = \begin{bmatrix} 0 & 0.685 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.9}$$

Equations 7.8 and 7.9 are in control canonical form where the coefficients of the characteristic polynomial are written in the first row of the matrix **A** (with opposite signs) and the coefficient(s) of the numerator polynomial are written in **C**.

7.2.1 State Feedback Control

As with classical control, feedback is used to control the behaviour of systems described in state-space. In state feedback designs the states **x** of the system are fed back (hence the name state feedback), and not the outputs of the system, as in classical control designs. This has the advantage that the amount of feedback of each of the dynamic elements (states) can be chosen arbitrarily, whereas in classical control designs only one feedback for the whole system can be chosen (one overall feedback in cascaded systems). Figure 7.2 shows a system with state feedback.

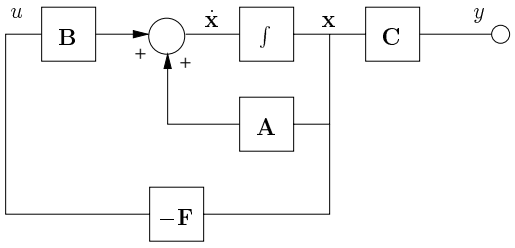


Figure 7.2: State feedback.

From the figure it can be seen that the feedback law is:

$$u = -\mathbf{F}\mathbf{x} \tag{7.10}$$

Inserting this in equation 7.1 gives:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}(-\mathbf{F}\mathbf{x}) = (\mathbf{A} - \mathbf{B}\mathbf{F})\,\mathbf{x} \tag{7.11}$$

which is the full state-space description of figure 7.2.

By choosing the values of $\mathbf{F}$ appropriately, the eigenvalues of the closed-loop can be determined.

To make a state feedback controller it must be ensured that the system is controllable.

7.2.2 Observer Design

An observer is used to estimate some or all of the states of a process that cannot possibly or easily be measured. Designing an observer is mostly a cheap and efficient way to obtain information about the system.

A full-order observer estimates all the states of a process, whereas a reduced-order observer only estimates some of them (usually those that cannot be measured otherwise). Although a reduced-order observer is slightly simpler, a full-order observer has the great advantage that the final closed-loop controller is less sensitive to sensor noise [fc, p 527].

As the observer will never make a precise estimate of the system's true states, the difference between the estimate and the true states is fed back to the observer. If the feedback gain $\mathbf{K}$ is sufficiently large, in comparison to the overall feedback gain, the observer error will be reduced quickly and the estimated states can then be used as a good substitute to the real states.

Figure 7.3 shows a full-order observer where an estimate $\hat{\mathbf{x}}$ of all the states of the process is given.

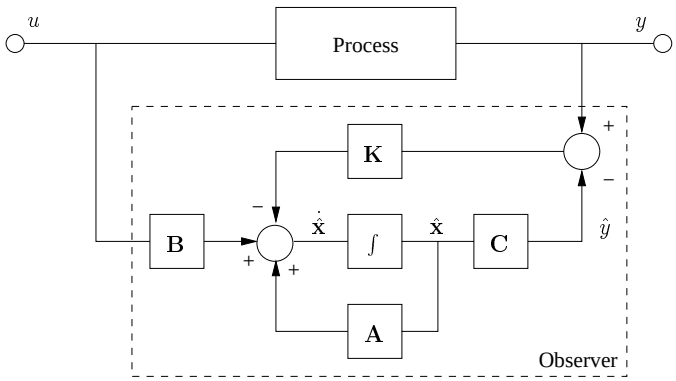


Figure 7.3: A full-order observer.

In the case of a SISO system, when the difference between the real output y and the estimated output $\hat{y}$ is fed back, the observer states can be described as:

$$\dot{\hat{\mathbf{x}}} = \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}u + \mathbf{K}(y - \mathbf{C}\hat{\mathbf{x}}) \tag{7.12}$$

where the feedback matrix is given by:

$$\mathbf{K} = \begin{bmatrix} k_1 \\ k_2 \\ \vdots \\ k_n \end{bmatrix} \tag{7.13}$$

The characteristic polynomial of the closed-loop observer is given by [fc, p 516]:

$$det\left[s\mathbf{I} - (\mathbf{A} - \mathbf{KC})\right] = 0 \tag{7.14}$$

Comparing the roots of this polynomial with the desired observer poles, the values of the feedback matrix $\mathbf{K}$ can be determined. An alternative and easier way to determine $\mathbf{K}$ is to use the MATLAB^{TM} command `place(A,C,Po)` where `A` and `C` are the matrices $\mathbf{A}$ and $\mathbf{C}$ and `Po` is a vector containing the desired observer poles.

Before designing an observer it must be ensured that the system is observable.

Integral Control

In integral control designs the integral of the error e (the difference between the outputs y and the reference signals y_r), as well as the states of the system $\mathbf{x}$, are fed back to the control signals u (see figure 7.4). Like in classical control

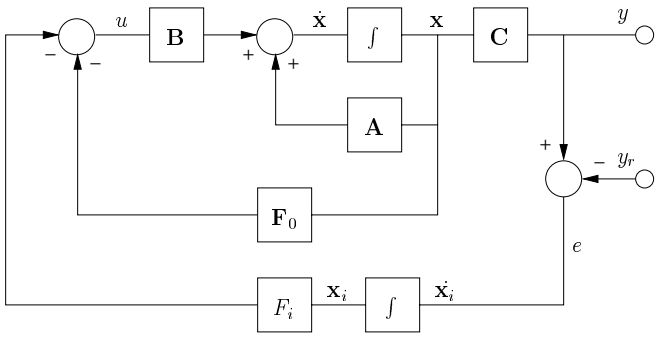


Figure 7.4: An integral controller.

designs, this gives a zero steady-state error and introduces an extra pole p_i.

The matrix $\mathbf{F}_0$ and the scalar F_i are the feedback gains and the integrator gain respectively. This yields that

$$\dot{x_i} = e = \mathbf{Cx} - y_r \tag{7.15}$$

Augmenting the state equations with equation 7.15 gives:

$$\begin{bmatrix} \dot{x_i} \\ \dot{\mathbf{x}} \end{bmatrix} = \begin{bmatrix} 0 & \mathbf{C} \\ \mathbf{0} & \mathbf{A} \end{bmatrix} \begin{bmatrix} x_i \\ \mathbf{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix} u - \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix} y_r \tag{7.16}$$

and the feedback law is:

$$u = -\begin{bmatrix} F_i & \mathbf{F}_0 \end{bmatrix} \begin{bmatrix} x_i \\ \mathbf{x} \end{bmatrix} = -\mathbf{F} \begin{bmatrix} x_i \\ \mathbf{x} \end{bmatrix} \tag{7.17}$$

The characteristic polynomial of the closed-loop system including the integral controller is given by:

$$det\left[s\mathbf{I} - (\mathbf{A}_i - \mathbf{B}_i\mathbf{F})\right] = 0 \tag{7.18}$$

where

$$\mathbf{A}_i = \begin{bmatrix} 0 & \mathbf{C} \\ \mathbf{0} & \mathbf{A} \end{bmatrix} \quad \text{and} \quad \mathbf{B}_i = \begin{bmatrix} 0 \\ \mathbf{B} \end{bmatrix} \tag{7.19}$$

As with the observer, the feedback gain matrix $\mathbf{F}$ can be found, using the MATLAB™ command `place`, but since the muliplication of the matrices $\mathbf{B}_i$ and $\mathbf{F}$ is in reverse order, the expression should be rewritten, using the fact that [fc, p 747]:

$$(\mathbf{A} - \mathbf{FB})^{\mathrm{T}} = \left(\mathbf{A}^{\mathrm{T}} - (\mathbf{FB})^{\mathrm{T}}\right) = \left(\mathbf{A}^{\mathrm{T}} - \mathbf{B}^{\mathrm{T}}\mathbf{F}^{\mathrm{T}}\right) \tag{7.20}$$

This means that the `place` command is `place(Ai',Bi',Pc)'` where `Pc` contains the poles of the closed-loop system and `'` denotes the transpose of a matrix. With the integrator an additional pole is added to the system and this must be chosen and placed in `Pc`.

7.3 State-space Description of the Compensator

To be able to implement the integral controller together with the observer (this configuration is called a compensator) on the PC, one total state-space model must be set up.

Figure 7.5 shows a block diagram of the total system, including the wind tunnel model.

From this it can be seen that:

$$\begin{aligned} u &= -\mathbf{F}_0\hat{\mathbf{x}} - F_i\mathbf{x}_i \\ \hat{y} &= \mathbf{C}\hat{\mathbf{x}} \end{aligned} \tag{7.21}$$

which gives:

$$\begin{aligned} \dot{\mathbf{x}}_i &= y - y_r \\ \dot{\hat{\mathbf{x}}} &= \mathbf{A}\hat{\mathbf{x}} + \mathbf{B}u + \mathbf{K}(y - y_r) \\ &= \mathbf{A}\hat{\mathbf{x}} - \mathbf{BF}_0\hat{\mathbf{x}} - \mathbf{B}F_i\mathbf{x}_i + \mathbf{K}y - \mathbf{KC}\hat{\mathbf{x}} \\ &= (\mathbf{A} - \mathbf{BF}_0 - \mathbf{KC})\hat{\mathbf{x}} - \mathbf{B}F_i\mathbf{x}_i + \mathbf{K}y \end{aligned} \tag{7.22}$$

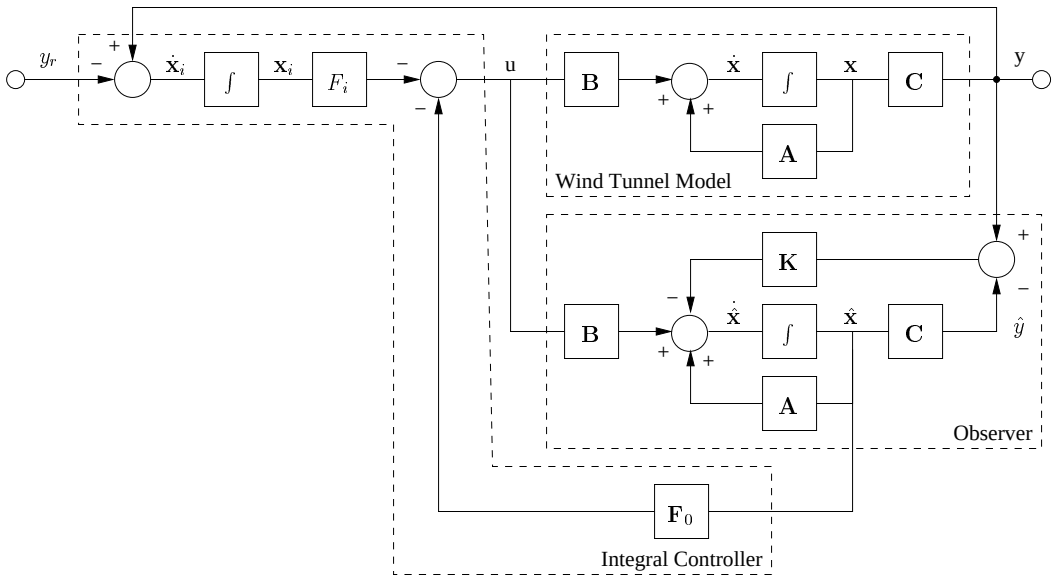


Figure 7.5: Block diagram of the full system.

From the states and their respective derivatives in equations 7.22 the state-space description of the compensator can be set up:

$$\begin{bmatrix} \dot{x}_i \\ \dot{\hat{\mathbf{x}}} \end{bmatrix} = \begin{bmatrix} 0 & \mathbf{0} \\ -\mathbf{B}\mathbf{F}_i & \mathbf{A} - \mathbf{B}\mathbf{F}_0 - \mathbf{K}\mathbf{C} \end{bmatrix} \begin{bmatrix} x_i \\ \hat{\mathbf{x}} \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ \mathbf{K} & \mathbf{0} \end{bmatrix} \begin{bmatrix} y \\ y_r \end{bmatrix}$$
$$u = \begin{bmatrix} -F_i & -\mathbf{F}_0 \end{bmatrix} \begin{bmatrix} x_i \\ \hat{\mathbf{x}} \end{bmatrix} \tag{7.23}$$

7.3.1 Discretisation of the Compensator

In order to implement the compensator in a computer a discrete-time model is needed. The following shows the manual procedure for converting a continuous-time state-space description to a difference equation that can be implemented in a computer.

Equations 7.1 describe the differential equations for a system and the solution to the two equations is [dc, p 105]:

$$\mathbf{x}(t) = e^{\mathbf{A}(t-t_0)}\mathbf{x}(t_0) + \int_{t_0}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}u(\tau)d\tau \tag{7.24}$$

which is a continuous-time equation. To come up with a discrete-time representation, a sample is used to get a difference equation. If $t = kT + T$ and $t_0 = kT$, the solution is:

$$\mathbf{x}(kT+T) = e^{\mathbf{A}T}\mathbf{x}(kT) + \int_{kT}^{kT+T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}u(\tau)d\tau \tag{7.25}$$

This equation still has the continuous time history $u(\tau)$ from the sample interval because the equation is not dependent on the type of hold. With zero order hold (ZOH) $u(\tau)$ is a constant in the sample interval:

$$u(\tau) = u(kT) \quad \text{for} \quad kT \leqq \tau < kT + T \tag{7.26}$$

and to simplify equation 7.25:

$$\eta = kT + T - \tau \tag{7.27}$$

Applying ZOH to equation 7.25 gives:

$$\mathbf{x}(kT + T) = e^{\mathbf{A}T}\mathbf{x}(kT) + \left(\int_0^T e^{\mathbf{A}\eta} d\eta\right)\mathbf{B}u(kT) \tag{7.28}$$

The matrices $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ are introduced to simplify equation 7.28 where

$$\mathbf{\Phi} = e^{\mathbf{A}T} \tag{7.29}$$

and

$$\mathbf{\Gamma} = \left(\int_0^T e^{\mathbf{A}\eta} d\eta\right)\mathbf{B} \tag{7.30}$$

The difference equations on standard form can be reduced to

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{\Phi}\mathbf{x}(k) + \mathbf{\Gamma}u(k) \\ y(k) &= \mathbf{C}\mathbf{x}(k) + \mathbf{D}u(k) \end{aligned} \tag{7.31}$$

Taking the z-transform of equation 7.31 with $\mathbf{D} = 0$ gives:

$$\begin{aligned} (z\mathbf{I} - \mathbf{\Phi})\mathbf{X}(z) &= \mathbf{\Gamma}U(z) \\ Y(z) &= \mathbf{C}\mathbf{X}(z) \end{aligned} \tag{7.32}$$

and the discrete transfer function is:

$$\frac{Y(z)}{U(z)} = \mathbf{G}(z) = \mathbf{C}(z\mathbf{I} - \mathbf{\Phi})^{-1}\mathbf{\Gamma} \tag{7.33}$$

The last step towards a discrete-time implementable model is to take the inverse z-transform of the transfer function of equation 7.33 and write it on recursive form.

In MATLAB™ the matrices $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ can easily be calculated with the function `c2d(ss_sys,Ts,'zoh')` where `ss_sys = ss(A,B,C,D)` is the state-space representation of the compensator, `Ts` is the sample time and `'zoh'` specifies that a zero-order-hold sampling is to be used. In addition to that the command `tf(sysd)` can produce the discrete-time tranfer function. The argument `sysd` is the discrete-time state-space description given by `c2d`.

7.4 Choice of Pole Locations

The three closed-loop poles `Pc` have been chosen by implementing the discrete-time compensator described by equations 7.40 and 7.41 (they will be examined later) in SIMULINK^{TM} and iteratively selecting values of `Pc`, until the closed-loop step-response has the desired overshoot, rise-time and settling time. The values of `Pc` are:

$$\mathrm{Pc} = -\begin{bmatrix} 8 & 9+7j & 9-7j \end{bmatrix} \tag{7.34}$$

For the step-up transfer function this results in a rise time t_r of 0.296 s, a settling time t_s of 0.784 s and zero overshoot. For the step-down transfer function this results in a rise time of 0.298 s, a settling time of 0.575 s and an overshoot M_p of 0.5%. The values have been found in SIMULINK^{TM}.

The reason why the choice of a set of complex conjugate poles results in a very small overshoot is that one state of the system consisting of VLT, motor and wind tunnel has poor observability. The two states are assumed to be the air acceleration (x_1) and the air velocity in the test section (x_2).

The poor observability is associated with x_1 which is not actually measured. This, in turn, means that oscillations in the air acceleration caused by complex conjugate poles, are only faintly visible on the air velocity in the test section. A conclusion to this is that the stagnation box works as intended.

The choice of real poles would result in a discrete-time step-response with a long rise time or a shape as shown in figure 7.6.

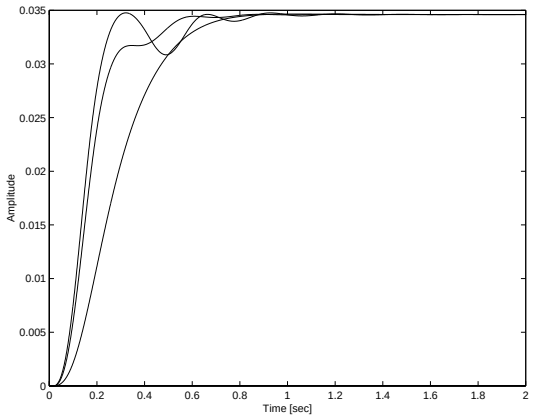


Figure 7.6: Step responses as a result of using real closed-loop poles.

Since the estimated state vector $\hat{\mathbf{x}}$ is fed back through the gain matrix F_0, the two observer poles `Po` should be faster than the two fastest closed-loop poles of the whole system by a factor 2–6 [fc, p 525]. Hence, the observer poles have been chosen as:

$$\mathrm{Po} = -\begin{bmatrix} 36+28j & 36-28j \end{bmatrix} \tag{7.35}$$

Using `place(Ai',Bi',Pc)'` with the values of `Pc` and:

$$\mathbf{A}_i = \begin{bmatrix} 0 & 0 & 0.685 \\ 0 & -6.44 & -19.8 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{B}_i = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{7.36}$$

with reference to equation 7.19, the closed-loop gain matrix $\mathbf{F}$ is:

$$\mathbf{F} = -\begin{bmatrix} F_i & \mathbf{F}_0 \end{bmatrix} = -\begin{bmatrix} 1518.2 & 19.6 & 254.2 \end{bmatrix} \tag{7.37}$$

Likewise, the observer feedback $\mathbf{K}$ has been computed with `place(A,B,Po)`:

$$\mathbf{K} = -\begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = -\begin{bmatrix} 2391.2 \\ 95.7 \end{bmatrix} \tag{7.38}$$

7.4.1 Computing the Compensator Matrices

Inserting the calculated values of $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{F}$ an $\mathbf{K}$ in the state-space description of the compensator given in equations 7.23 gives:

$$\begin{aligned} \begin{bmatrix} \dot{x}_i \\ \dot{\hat{x}}_1 \\ \dot{\hat{x}}_2 \end{bmatrix} &= \begin{bmatrix} 0 & 0 & 0 \\ 1518.2 & -26 & -1912 \\ 0 & 1 & -65.56 \end{bmatrix} \begin{bmatrix} x_i \\ \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} + \begin{bmatrix} 1 & -1 \\ 2391.2 & 0 \\ 95.7 & 0 \end{bmatrix} \begin{bmatrix} y \\ y_r \end{bmatrix} \\ u &= \begin{bmatrix} -1518.2 & -19.6 & -254.2 \end{bmatrix} \begin{bmatrix} x_i \\ \hat{x}_1 \\ \hat{x}_2 \end{bmatrix} \end{aligned} \tag{7.39}$$

which is a system with two inputs y and y_r and one output u.

With the use of the command `c2d` in conjunction with `tf` as mentioned earlier, the two discrete-time tranfer functions describing the compensator can be found:

$$\frac{U_1(z)}{Y_r(z)} = \frac{25.25z^{-1} - 20.7z^{-2} + 5.781z^{-3}}{1 - 1.569z^{-1} + 0.7294z^{-2} - 0.1602z^{-3}} \tag{7.40}$$

and

$$\frac{U_2(z)}{Y(z)} = \frac{-615.9z^{-1} + 1083z^{-2} - 477.6z^{-3}}{1 - 1.569z^{-1} + 0.7294z^{-2} - 0.1602z^{-3}} \tag{7.41}$$

These have been used to find the closed-loop poles, mentioned before.

Taking the inverse z-transform of either of the two tranfer functions and writing these on recursive form gives the difference equations:

$$\begin{aligned} u_1[n] = {} & 25.25y_r[n-1] - 20.7y_r[n-2] + 5.781y_r[n-3] + 1.569u_1[n-1] \\ & - 0.7294u_1[n-2] + 0.1602u_1[n-3] \end{aligned} \tag{7.42}$$

and

$$u_2[n] = -615.9y[n-1] + 1083y[n-2] - 477.6y[n-3] + 1.569u_2[n-1] - 0.7294u_2[n-2] + 0.1602u_2[n-3] \tag{7.43}$$

The total control signal is obtained by adding together the two control signals given by equations 7.42 and 7.43 such that $u[n] = u_1[n] + u_2[n]$.

7.5 Simulation of the State-space Controller

To verify the design of the state-space compensator, a number of simulations have been made using MATLAB™ and SIMULINK™. The details of these simulations are given in appendix A.2 and the outline of the tests are listed below:

- Step response without any disturbances.
- The fluctuations of the control signal, i.e. the control effort.
- Step response with sine disturbances on u and the feedback path respectively.
- Step response with random signals on u and feedback path respectively.
- Response to a ramp input.

All step responses have been made with a step reference input from 0 to 0.0346, resulting in a step from 0 to 1 V on the control signal u. The frequencies of the sine signal are 5 Hz, 50 Hz and 1 kHz as in the simulation of the PID controller.

The influence of the different noise signals on the output of the closed-loop system are summarized in table 7.1.

It can be seen that noise applied to the feedback path can result in great disturbances on the ouput, whereas noise on the control signal has little or no influence. This sensitivity could be reduced by selecting slower closed-loop poles which, on the other hand, would result in a slower compensator.

The ramp input is a ramp with a slope of 0.0346 so that the input reaches its maximum (0.346) after 10 seconds in accordance to the limits of the setpoint changer (see chapter 5 on page 34). It produces a tracking error of maximum 0.009 when the setpoint changer changes a setpoint with the maximum slope of 0.0346.

Noise Signal	Feedback	Control Signal
5 Hz sine	Visible*	Visible*
50 Hz sine	Visible*	None
1 kHz sine	Visible*	None
Random ($\frac{signal}{100,000}$)	Visible*	-
Random ($\frac{signal}{10,000}$)	Dominant*	-
Random ($signal \times 2$)	-	None*
Random ($signal \times 20$)	-	Visible*

Table 7.1: The effects of noise on the output signal. An (*) indicates that the control signal is varying dramaticly.

For a step of 1 V on the control signal, the peak control effort is about 2.5 V. When the controller regulates in the vicinity of it upper limit (10 V), this means that the control signal will saturate, resulting in a slower response (a rise time of approximately 0.49 s).

7.6 Test of the State-space Controller

The state-space controller has been tested by implementing the difference equations 7.42 and 7.43 in a control software that will be described in chapter 10 on page 77. The tests are described in appendix F.

As with the test of the two PID controllers, the state-space controller did not behave as expected when installed in the tunnel system. It required some coarse adjustment of the poles to make the controller function just tolerably.

The closed-loop poles after the adjustment are placed in -3, -4 and -4.1 and the observer poles in -10 and -11 which obviously results in a slower system.

This drowsing of the system was chosen to prevent the VLT from going into current saturation as a consequence of the wrong model due to the falsely calibrated sensors.

The rise time of the measured system is approximately 1 s, the overshoot is 30% and the steady state error does not come within the $\pm 1\%$ limit at all.

Even after the detuning the compensator showed a slightly oscillating behaviour.

7.7 Conclusion

The simulated closed-loop system, consisting of a state-space compensator and the wind tunnel model, has the following properties:

System	t_r [s]	t_s [s]	M_p [%]	ω_b $[\frac{\text{rad}}{\text{s}}]^*$	e_{ss} step	$e_{ss,\max}$ ramp
$G_u(z)$	0.308	0.784	0	6.24	0	0.0091
$G_d(z)$	0.302	0.575	0.5	6.21	0	0.0091

Table 7.2: The features of the simulated closed-loop system consisting of the state-space compensator and wind tunnel model. *) The bandwidth has been measured for the continuous-time system in `ltiview`.

$G_u(z)$ and $G_d(z)$ are the closed-loop step responses with the step-up and step-down transfer functions of the tunnel respectively. They are shown on figure 7.7 where $G_u(z)$ is the solid line and $G_d(z)$ is the dotted line.

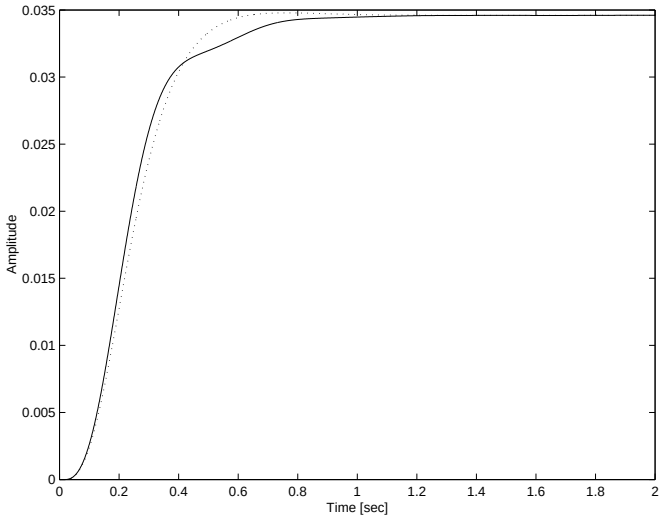


Figure 7.7: Response to a step-up (solid line) and a step-down signal (dotted line), respectively.

From table 7.2 it can be seen that the simulated state-space compensator meets most of the requirements of the demand specification, namely:

- Resulting closed-loop bandwith $\omega_b < 15$ $\frac{\text{rad}}{\text{s}}$.
- Allowable overshoot $M_p = \pm 1\%$.
- $\pm 1\%$ state error for a step input.

The requirement of a rise time t_r of approximately 0.2 s has not been met. A longer rise time was accepted to have a more consistent response all over the control signal range (0–10 V). Otherwise the rise time for setpoints close to the lower and upper limit of the signal range would vary quite much.

The measurements in the tunnel show somewhat other results, due to model deviations. The poles of the state-space compensator has been detuned to

attain acceptable results. However, the undesired oscillating behaviour makes it a rather unsuitable solution for the final design.

With a better model of the wind tunnel system, including motor/fan and VLT, it would be posible to make a better state-space compensator.

Chapter 8

Module 4 - Choice of Controller

Through chapters 6 and 7 three different controllers have been designed for the wind tunnel. The purpose of this chapter is to compare the three controllers and pick the best suited controller for the wind tunnel. The controllers will be compared with regard to:

- Speed.
- Control effort.
- Overshoot.
- Noise rejection.
- Steady state / tracking error.

The comparison will be done in three ways: by regarding the key features from the simulations, by regarding the key features from the measurements of the controllers implemented in the tunnel system and with the use of a formalized cost function (LQ).

8.1 Comparing the Simulated Features

The simulation data used for the comparison are shown in table 8.1.

Controller	Plant	t_r [s]	t_s [s]	ω_b $[\frac{\text{rad}}{\text{s}}]$	$e_{ss,\max}$ (ramp)	control effort
$\text{PID}_{\text{no filter}}$	$G_u(z)$	0.4	2.6	5.7	0.0135	+20%, -15%
	$G_d(z)$	0.5	1.8	4.4	0.0129	-
$\text{PID}_{\text{filter}}$	$G_u(z)$	0.7	4.9	3.5	0.0330	+0%, -15%
	$G_d(z)$	0.8	4.0	2.8	0.0325	-
State-sp.	$G_u(z)$	0.308	0.784	6.24	0.0091	150%
	$G_d(z)$	0.302	0.575	6.21	0.0091	-

Table 8.1: The simulated comparison data for the three controllers.

From the table it can be seen that the state-space compensator is the faster, but at the cost of a big control effort.

The PID controller with filter is obviously the slowest with a settling time of 4–5 s. This, on the other hand, results in less noise sensitivity and a smaller control effort and thus a less aggressive controller.

As the only one, the state-space compensator has an undesired overshoot when applied to the step-down function. However, this is only 0.5% which is within the specified limits of the demand specification.

All three controllers have a non-zero tracking error. This, however, is assumed not to be a big problem since the steepest possible ramp (resulting in the largest tracking error) only occurs when the setpoint changer is active. After that, the respective controllers will have to correct the (tracking) error listed in the table, which is relatively small.

When the controllers are set to produce a ramp output, the error will be smaller than the value found in the table. If the controller is set to change the ouput value from 0 to maximum over 5 minutes (300 s) e.g., the ramp error will be only $\frac{10}{300}$ of the table value.

8.2 Comparison Using a Cost Function

LQ optimization is a way to choose the optimum compromise between performance and control effort of a controlled system. That is to minimize the performance index $\mathcal{J}$ given by:

$$\mathcal{J} = \frac{1}{N}\sum_{n=0}^{N}\left((y[n] - y_{ref}[n])^2 + \rho\,(u[n] - u[n-1])^2\right) \tag{8.1}$$

The used symbols is explained in 8.2.

Symbol	Explanation
N	The number of samples in a measured or simulated step response
$y[n]$	The actual magnitude of the nth output sample
$y_{ref}[n]$	The desired magnitude of the nth output sample
$u[n]$	The actual magnitude of the nth control signal sample
ρ	weighting factor between control effort and output error.

Table 8.2: The used symbols in the LQ optimization.

The performance index gives a quantitative measure of the mean-square value of the fluctuations of u and y and a smaller value of $\mathcal{J}$ yields a "better" system.

8.2.1 Performance Index of the Simulated Controllers

Two performance indeces have been calculated for each of the three simulated controllers: one with $\rho = 1$ and one with $\rho = 0.1$. In the first case the control effort and output error have been weighted equally whereas, in the latter case, the output error is considered more important than the control effort.

The computations of $\mathcal{J}$ have been made in MATLAB™ and SIMULINK™ and the results are listed in table 8.3. In the first case where ρ is 1, the PID

Controller	Response	$\mathcal{J} \times 10^4$ ($\rho = 1$)	$\mathcal{J} \times 10^5$ ($\rho = 0.1$)
$\text{PID}_{\text{no filter}}$	$G_u(z)$	2.4937	4.9037
	$G_d(z)$	2.5264	5.2839
$\text{PID}_{\text{filter}}$	$G_u(z)$	0.7899	7.4795
	$G_d(z)$	0.8616	8.1710
State-sp.	$G_u(z)$	4.7759	6.6819
	$G_d(z)$	4.1103	6.1293

Table 8.3: Performance indeces for the three simulated controllers.

controller with filter shows best results. When, in the case where ρ is 0.1, the output accuracy is considered most important, the PID controller without filter shows the best performance.

8.2.2 Performance Index of the Measured Controllers

As with the simulated controllers, two performance indeces have been calculated for each of the three measured closed-loop controllers: one with $\rho = 1$ and one with $\rho = 0.1$ and the results are shown in table 8.4.

Controller	Plant	$\mathcal{J}$ ($\rho = 1$)	$\mathcal{J}$ ($\rho = 0.1$)
$\text{PID}_{\text{no filter}}$	Step	0.1925	0.1905
	Ramp	78.348	78.340
$\text{PID}_{\text{filter}}$	Step	0.2039	0.2039
	Ramp	81.666	81.665
State-sp.	Step	0.1558	0.1550
	Ramp	69.823	69.826

Table 8.4: Performance indeces for the three measured closed-loop controllers. The ramp and step inputs include both a positive and a negative ramp and step respectivly.

The numbers for the PID without filter and the state-space controller apply to the adjusted controllers. From this it can be seen that the state-space controller gives best results both with $\rho = 1$ and $\rho = 0.1$.

8.3 Choice of Controller

To make the choice of a controller the main consideration is how the controllers perform with the tunnel. The key figures from tables 8.1 through 8.4 are used as guidelines.

From the test of the controllers (see appendix F) it can be seen that the PID controller witout filter and the state-space controller needed to be adjusted to work satisfactory. After the adjustments the PID without filter has a $t_r = 1.8$ s, the PID with filter has a $t_r = 4$ s and the state-space has a $t_r = 1$ s. This adjustments resulted in the state-space controller having the best performance index of the three controllers.

Comparing figures F.2 to F.7 it can be seen that the state-space controller and PID controller without filter produces an overshoot a step input as well as a ramp input. The overshoot with a step input is approximately 30% and 20% for the state-space and PID respectivly. The PID with a filter does not produce this overshoot, but it makes a tall spike in the output at the end of the ramp. This spike is assumed to be the consequence of the large tracking error that the controller produces. This is visible on the graphs and it also results in an error message on the VLT, that has to be manually reset. This is the main reason that the PID with filter is not implemented.

The chosen controller is therefore the PID without filter. The reason is that this controller gives the best performance in the tunnel.

Chapter 9

Module 5 - Hardware Configuration

This module describes the connections between the PC, the transducers and the VLT. All practical initiatives to be done to make everything work together is described briefly. At first the transducer couplings are descibed, afterwards the VLT setup and finally the connections to the I/O-card.

9.1 The Pressure Transducers

The transducers used to measure the air velocity inside the tunnel are described in the analysis in section 2.6. Nine pitot tubes at the end of the test section are used to measure the pressure inside the tunnel. The nine pitot tubes are mounted as shown in figure 9.1. The horizontal tubes are connected to each other so that each pair of transducers measures the mean pressure on the three interconnected tubes.

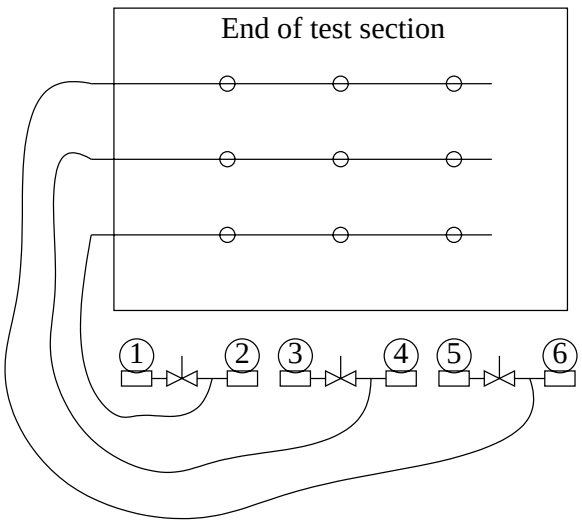


Figure 9.1: Connection of transducers and pitot tubes at the end of the test section.

As shown in figure 9.1 six transducers are coupled in pairs to measure the pressure in the wind tunnel. One of the two is calibrated to measure pressure at the air velocities from 0 $\frac{\text{m}}{\text{s}}$ to ≈ 14 $\frac{\text{m}}{\text{s}}$ and the other is calibrated to measure pressure at air velocities up to ≈ 25 $\frac{\text{m}}{\text{s}}$. A solenoid valve removes the pressure from the low pressure transducers when the pressure exceeds 14 $\frac{\text{m}}{\text{s}}$. The valve is controlled by a digital output from the I/O-card. The digital output comply with the TTL standard, which means the logical high output V_{OH} is between 2.7 and 5 V. Since the switch needs 12 V to turn on a, non-inverting amplifier curcuit has been designed to amplify the output.

A diagram for the control of the valves is shown in figure 9.2

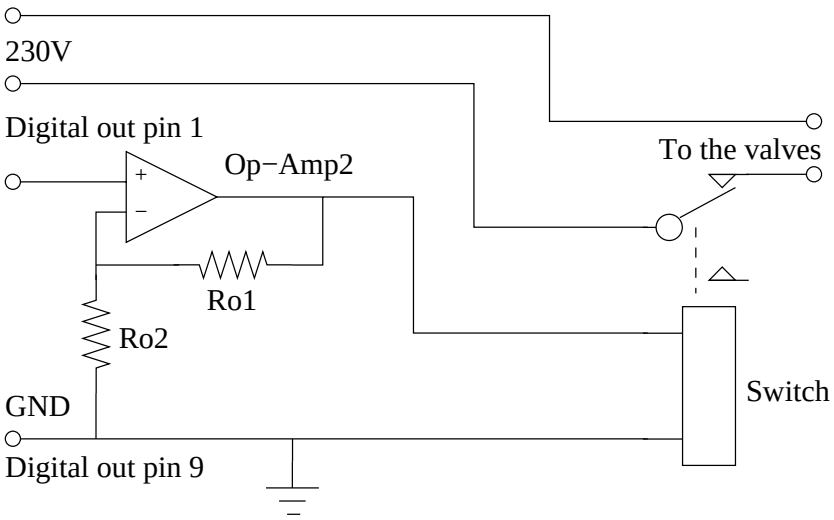


Figure 9.2: The control of the valves.

Table 9.1 shows the measuring range of each transducer. Transducer 1, 3 and 5 measure the low pressures and transducer 2, 4 and 6 measure the high pressures.

The sample software converts the measured pressure to an air velocity with the use of formula 9.1:

$$v = \sqrt{\frac{2 \cdot p_d}{\rho_{air}}} \tag{9.1}$$

where the air density $\rho_{air} \approx 1.20$ $\frac{\text{kg}}{\text{m}^3}$ at $20°C$. The calibrated maximum air velocity for each transducer is shown in table 9.1.

Table 9.1 shows that the maximum usable air velocity measurement is 24.8 $\frac{\text{m}}{\text{s}}$. It was not possible to install pressure transducers that could measure the range from 0 $\frac{\text{m}}{\text{s}}$ to 30 $\frac{\text{m}}{\text{s}}$.

The outputs from the transducers are their supply current. The range is 4–20 mA linearly, which means that they use 4 mA when no air velocity is mea-

Transducer number	Max. pressure	Max. air velocity
1	118.3 Pa	14.0 $\frac{\mathrm{m}}{\mathrm{s}}$
2	369.7 Pa	24.8 $\frac{\mathrm{m}}{\mathrm{s}}$
3	121.6 Pa	14.2 $\frac{\mathrm{m}}{\mathrm{s}}$
4	490.3 Pa	28.6 $\frac{\mathrm{m}}{\mathrm{s}}$
5	176.5 Pa	17.1 $\frac{\mathrm{m}}{\mathrm{s}}$
6	630.6 Pa	32.4 $\frac{\mathrm{m}}{\mathrm{s}}$

Table 9.1: Transducer range and corresponding maximum air velocity.

sured and 20 mA when the maximum air velocity at each transducer is measured.

The I/O card used to sample the output is set to the maximum range from 0 V to 10 V, which is chosen to minimize the noise influence. A series resistance is placed on the transducer supply and the voltage over the resistance is sampled.

The resistance must be

$$R = \frac{U}{I} \Rightarrow R = \frac{10\ \mathrm{V}}{20\ \mathrm{mA}} = 500\ \Omega \tag{9.2}$$

To make sure the resistance is 500 Ω a potentimeter of 1 kΩ is placed in parallel with af resistance of 1 kΩ to calibrate the resistance to exactly 500 Ω.

According to the data sheet the supply voltage for each transducer must be within a range of 20 V to 55 V for a series resistance of 500 Ω according to the data sheet. A supply voltage of 30 V is chosen.

9.2 The Temperature Sensor

As mentioned in the analysis in section 2.6.2 the temperature sensor is a PT100 sensor. It has a resistance of 100 Ω at 0 °C and 120 Ω at 50 °C. To measure the small changes in resistance, a Wheatstone bridge and a differential amplifier is used. This is shown in figure 9.3. This amplifier must be placed near the sensor in order to make it less sensitive to electrical noise.

9.3 The VLT

The frequency converter is a Danfoss 6000 VLT. It must be set up in accordance with the equipment connected to it in order to control the air velocity inside the wind tunnel. The setup is shown in appendix E.

With this setup the frequency of the VLT can be controlled with an analogue input voltage between 0 V and 10 V on the input pins 53 and 55. The VLT can

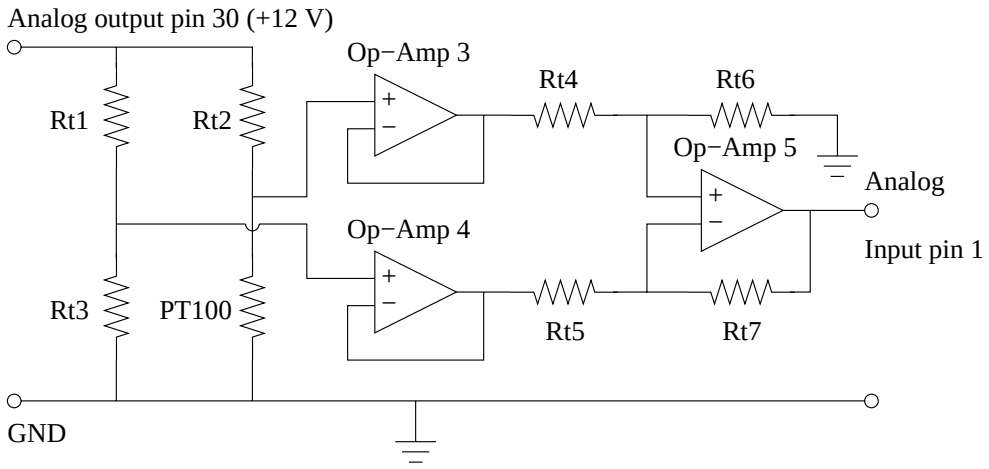


Figure 9.3: The temperature sensor curcuit.

be turned on and off according to the input voltage on pins 18 and 20. When the voltage exceed 10 V the VLT is turned on and below 5 V it is turned off. An amplifier like figure 9.2 is used to amplify the digital control signal.

9.4 The PC

The communication between the PC and the VLT and the sensors takes place through the I/O card. The I/O card is a NuDAQ PCI-9112 with 16 digital inputs and outputs, 8 differential analogue inputs and two analogue outputs. The temperature sensor is connected to analogue input 0 and the pressure transducers to input 1–6. The VLT control voltage is connected to the digital output 1 via a buffer and the VLT on/off signal is connected to the analogue output 0 through an amplifier.

A diagram for sensor 1 is shown in figure 9.4.

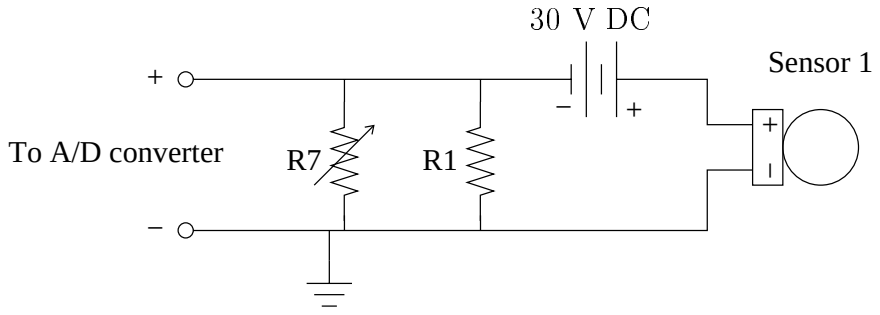


Figure 9.4: Diagram for sensor 1.

The complete diagram for the connection unit is shown in appendix I.

Chapter 10

Module 6 - Software

This chapter considers the design and implementation of the graphical user interface (GUI) and the controller algorithm. Through the GUI the application must have the following functions:

- A field to change the air velocity setpoint and a button to submit the air velocity.
- Fields to make the air velocity follow a user preset ramp and a button to submit the ramp.
- A menu where the data from a test series can be saved to a file.
- A menu where the user manual for the program can be accessed.
- A button to stop the system.

Furthermore, the GUI must contain the following information about the wind tunnel:

- The actual air velocity in the test section of the wind tunnel.
- The actual temperature in the test section of the wind tunnel.
- Ramp status. This option indicates how far the system is with a given ramp.
- An indicator to show when the air velocity has reached its desired value.

In order to handle the real time demands from the controller algorithm, the application must be separated into two processes: one handling the GUI and one controlling the air velocity in the wind tunnel. This is a consequence of the use of Windows® 2000 as the operating system and hereby the use of Win32 application programming interface (API). A description of the Win32 API can be found in appendix B.

Through a number of tests it has been concluded that the better way to establish a communication between the two processes is through a pipe. This

conclusion is made from the fact that the Windows® message system is too slow when messages are sent with 50 Hz. For more information about the Windows® message system and pipes refer to appendix B.

In order to handle this communication a protocol is defined.

10.1 Protocol

This protocol is defined so that the parameters shown in table 10.1 can be sent between the two processes.

Package identifier	Byte no.	Content
ID_SETPOINT	0	ID_SETPOINT
	1	Low byte of `Setpoint_New` times 10
	2	high byte of `Setpoint_New` times 10
ID_RAMP	0	ID_RAMP
	1	Sign of `RampIncremental`
	2	Low byte of `RampIncremental` times 10^8
	3	Mid byte of `RampIncremental` times 10^8
	4	High byte of `RampIncremental` times 10^8
	5	Low byte of `RampFrom` times 10
	6	High byte of `RampFrom` times 10
	7	Low byte of `RampTo` times 10
	8	High byte of `RampTo` times 10
ID_DATA_FROM_PROCESS	0	ID_DATA_FROM_PROCESS
	1	1. char of `ActualAirFlow`
	2	2. char of `ActualAirFlow`
	3	3. char of `ActualAirFlow`
	4	4. char of `ActualAirFlow`
	5	1. char of `ActualTemperature`
	6	2. char of `ActualTemperature`
	7	3. char of `ActualTemperature`
	8	4. char of `ActualTemperature`
	9	1. char of `RampStatus`
	10	2. char of `RampStatus`
	11	3. char of `RampStatus`
	12	4. char of `RampStatus`
	13	5. char of `RampStatus`
ID_CLOSE_PROCESS	0	ID_CLOSE_PROCESS

Table 10.1: The protocol defined to handle the communication between the GUI process and the controller process.

The protocol is defined this way, because all the data is handled in floats and since bit operations on floats are not possible with the used Borland 5.5 compiler.

The setpoint was given to have a maximum of one decimal. Therefore it is chosen to multiply it with 10, convert it into an integer, split it into two bytes and send these.

The same goes for the ramp. The ramp `From` and `To` options should have a maximum of one decimal. Therefore these are handled like the setpoint. The ramp `Time` option can consist of as many digits as needed, as long as the slope of the ramp is within 0.0000055555556 $\frac{\text{m}}{\text{s}\cdot\text{sample}}$ to 0.06 $\frac{\text{m}}{\text{s}\cdot\text{sample}}$. This ramp is given to have a minimum slope so that a ramp from 0 $\frac{\text{m}}{\text{s}}$ to 24 $\frac{\text{m}}{\text{s}}$ will take 24 hours. The steepest slope is given in the setpoint changer module.

When the actual air velocity, temperature and ramp status is sent from the controller process, it is chosen to convert these into characters before they are sent. This is done so that no further conversion is needed in the GUI process before these values can be shown on the screen.

10.2 GUI Design

The idea of the GUI is to give a user friendly interface to the wind tunnel. The purpose is to implement the user facilities given in the demand specification (and those pointed out in the beginning of this chapter).

In order to handle the user inputs on the screen and communicate with the controller process simultaneously, the GUI process is built up around two threads; a main thread that handles the users input and a buffer thread that handles the communication with the controller process. This configuration is shown i figure 10.1.

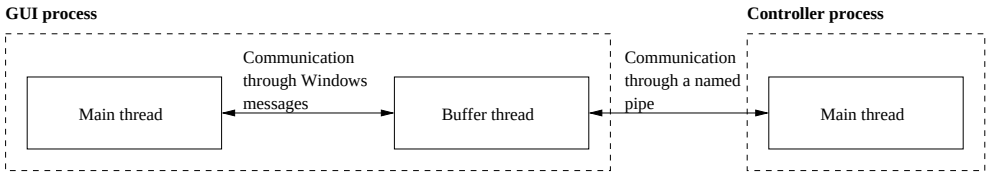


Figure 10.1: The organization of the GUI process and how it interacts with the controller process.

The communication between the buffer thread and the controller process is time critical, because the buffer thread has to work as a temporary buffer that saves the data recieved from controller process. The communication between the main thread and the buffer tread is not time critical, since the buffer thread has been chosen only to send information to the GUI process every 2 seconds.

10.2.1 Main Thread

First step is to create the main thread. A flow chart for the main thread is shown in figure 10.2.

The first thing the main thread has to do is to make an initialization. The

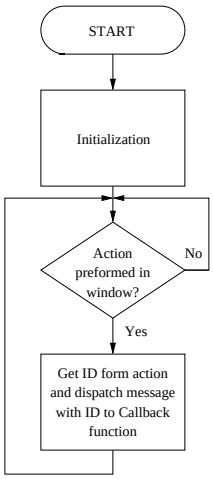


Figure 10.2: Flow Chart of the main thread.

things done in this initialization are listed below:

Creation of main program window: The empty main window is drawn. The only options set up for the main window are: the menu bar, how the icon for the .exe file looks and how the mouse cursor looks in the window. The only reason that the main window is created in the initialization phase (in the `main` function) is that it is the only way to initialize a Windows® message queue. Windows® message queues are necessary to make a graphical Windows® application work.

Creation of the buffer thread: The buffer thread that has to communicate with the controller process is created.

Creation of the controller process: The controller process is started with the highest Windows® 2000 priority (`REAL_TIME_PRIORITY_CLASS`).

After the initialization the main thread goes into a loop that checks if an action has occurred in the window and dispatches the message associated with the action recieved. The message is recieved by the callback loop and the function linked to the message is executed. For more information about the callback loop see appendix B.

The next step is to create the graphical outline according to the demand specification. This is done by adding buttons, listboxes and editboxes to the main window by using the Win32 API function `CreateWindow` (see appendix B). The drawing of the window is carried out, when the Windows® kernel sends the `WM_CREATE` message. This message is send from the kernel just after the message queue has been created. The result when the GUI has been drawn can be seen in figure 10.3.

Afterwards, functions to handle the editboxes, listboxes, buttons and the menu

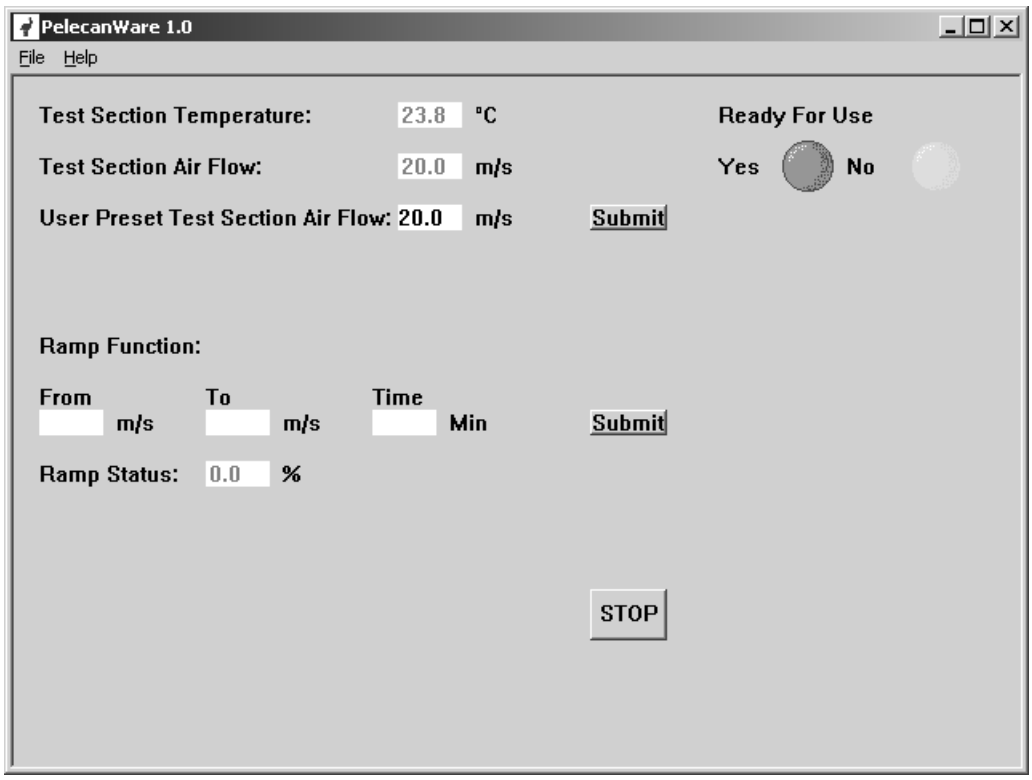


Figure 10.3: The outline of the GUI.

bar is drawn on the screen are assigned. From the demand specification the functionality of the GUI was given and is shown below for convenience:

The Air Velocity `Submit` button: Sends the desired air velocity written in the air velocity editbox to the buffer thread which passes it to the controller process. Before the desired air velocity is sent, a function checks if the data is valid. Valid data has a range of 0 to 24 $\frac{\text{m}}{\text{s}}$ and can consist of decimal numbers. If an invalid character is written an error message appears.

The ramp `Submit` button: Checks whether the values in the three editboxes `From`, `To` and `Time` are valid. The valid values for the `From` and `To` editboxes are the same as for the test section air velocity. For the `Time` editbox the `Submit` button only tests if a valid type is entered, not the range of the number. Finally, the slope of the ramp is tested. If invalid data is detected messages will appear on the screen. A maximum of 1440 min can be entered which is approximately 24 hours.

The `STOP` button: Terminates the current action by sending an air velocity value of 0 $\frac{\text{m}}{\text{s}}$ to the controller process through the buffer thread.

The menubar:

File: Has two submenus: `save` and `exit`. When `save` is selected a dialogbox appears and a desired location for saving the temporary log data can be selected. The `exit` sub menu terminates the application by sending a `WM_CLOSE` command (see appendix B).

Help: When selected, a dialogbox with the help screen appears.

The last thing needed is the indicator for the wind tunnel. This is made by switching two bitmap pictures (a red and a green) between transparent and visible.

The drawing of the bitmaps is done whenever a `WM_PAINT` message is sent. This message is sent from the kernel whenever the window is resized or moved.

To determine which of the two pictures to be visible, and in order to keep the use of Windows® messages to a minimum, the buffer thread compares the actual air velocity with the setpoint and set a variable (`ready`) according to the situation found. This `ready` variable is send to the main thread every 2 s in a `WM_PAINT` message with the `ready` variable in the last of the three parameters in a Windows® message. The parameter is 2 if the tunnel is ready and 1 if the tunnel is not ready.

10.2.2 Buffer Thread

According to the demand specification a log file with the collected data should be made, and saved in a desired location. A temporary collection of data is made in the thread as it receives data from the controller process. The data is saved in an array, and when the data of 100 cycles is received a pointer to the data is sent to the main thread. This is done to keep the data flow and the disk access down to a minimum. A flow chart of the buffer thread can been seen on figure 10.4.

The buffer thread also checks if the tunnel is ready by comparing the actual air velocity received from the controller process with the desired setpoint from the user. In the demand specification it was given that an error of $\pm 1 \frac{\mathrm{m}}{\mathrm{s}}$ of the actual air velocity is acceptable. If the difference between the actual air velocity and the desired setpoint is within this range the variable `ready` is set to 2 and otherwise to 1.

Before the `ready` variable is sent, a pointer to the actual air velocity, temperature and ramp status characters (received from the controller process) is sent to the main thread in a Window message with a defined identifier `ID_UPDATE_GUI`.

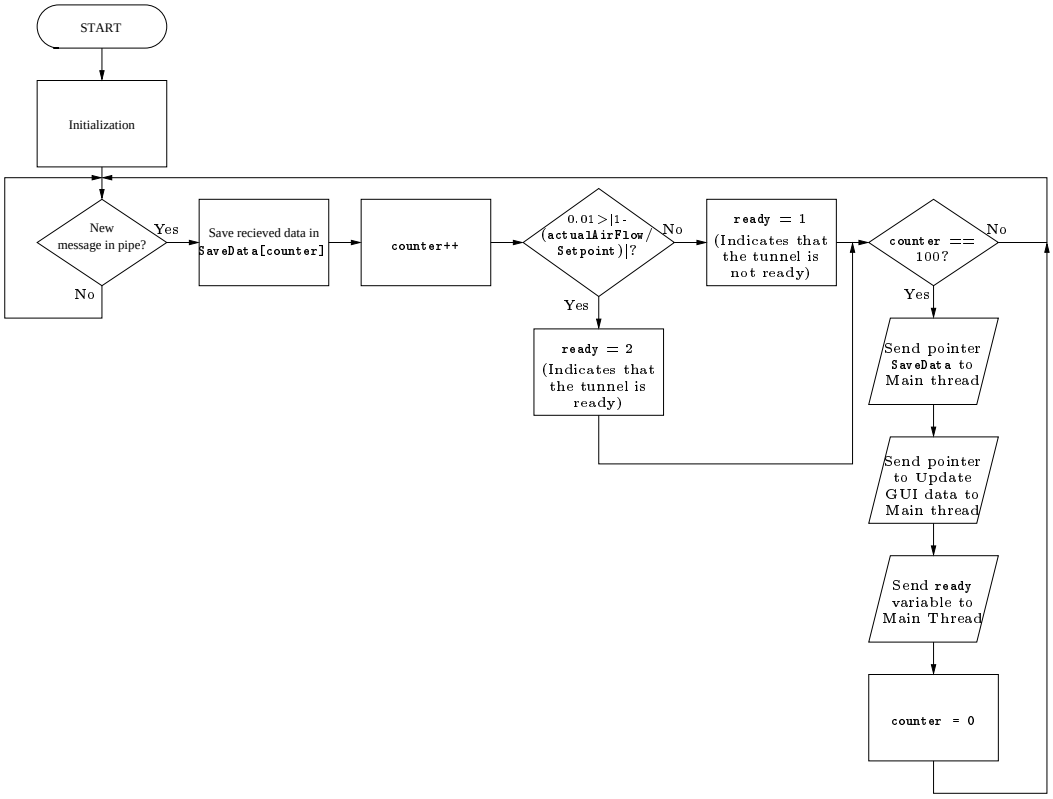


Figure 10.4: Flow Chart of the buffer thread.

10.3 Controller Process Design

The purpose of the controller process is to control the air velocity in the wind tunnel. The process has to control the sample rate, determine the actual air velocity and temperature in the wind tunnel, calculate the next control signal and send data to and receive data from the GUI process. In figure 10.5 a flow chart of the controller process is shown.

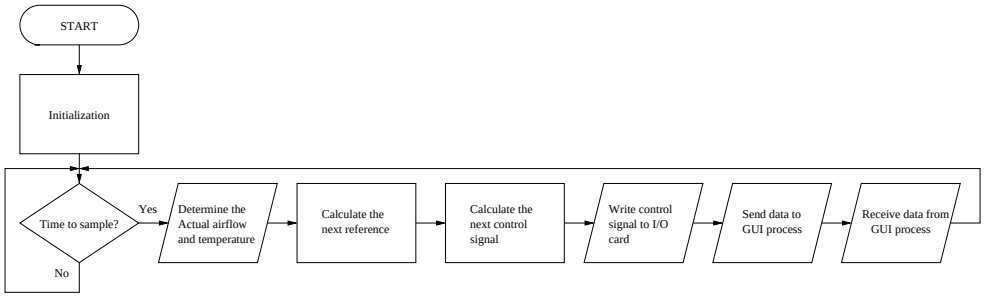


Figure 10.5: The overall loop of the controller process.

In the following sections each step of the flow chart is explained.

10.3.1 Initialization

The initialization phase does the following:

Initialize the data acquisition card: The data acquisition card needs to be initialized before it can be used. This is done with the `RegisterCard` function written in the device driver for the I/O card [cdrom].

Connect to the pipe: In order to communicate with the GUI process through a pipe, the controller process needs to connect to the pipe. For the connection to work, the controller has to connect with the same attributes as when the pipe is created in the GUI process, that is: byte oriented messages, same size of in- and output buffers (5000 bytes each) and read and write access attributes.

Create a timer queue: In order to have a fixed sample rate a timer is needed. In the Windows® 2000 kernel, there is only one usable way to create this timer; by using the `CreateTimerQueueTimer`. This function will then generate a software interrupt and call a specific callback function with a specified frequency. The `CreateTimerQueueTimer` function can generate interrupts up to 100 Hz which is adequate for this purpose since only a sample frequency of 50 Hz is needed.

When the initialization is completed the main loop can begin: Whenever the `CreateTimerQueueTimer` generates an interrupt and calls the callback function, the tasks described in the following sections will be run.

10.3.2 Determine the Actual Air Velocity and Temperature

In order to determine the actual air velocity and temperature this function needs to read data from the data acquisition card. This is done by reading the seven A/D converter registers where the readings from the six pressure sensors and the temperature sensor are stored. A reading of the pressure is a voltage proportional with the pressure in $\mathrm{mmH_2O}$.

From the six pressure values the function has to determine which three to use and calculate the mean pressure from the selected sensors. The six sensors have different pressure spans, where three of the sensors are calibrated to measure low pressures with high accuracy, and three sensors cover the entire area. When shifting between the sensors a hysteresis area is made to prevent it from disturbing the measurements, when operating in the shifting area.

The pressures are measured in $\mathrm{mmH_2O}$ and are converted to Pa with the following equation:

$$P_{\mathrm{Mean}} = 9,80665 \cdot P_{\mathrm{mmH2O}} \tag{10.1}$$

P_{Mean} is then converted to the actual air velocity with the following equation:

$$v_{\text{Air}} = \sqrt{\frac{2 \cdot P_{\text{Mean}}}{\rho_{\text{Air}}}} \tag{10.2}$$

where ρ_{Air} is the air density at the given temperature.

The voltage read for the temperature is converted to the actual temperature and returned. The conversion is done with the equation:

$$T_{\text{Air}} = 5 \cdot V_{\text{Read}} \tag{10.3}$$

10.3.3 Calculate the next Reference

Before the next control signal can be calculated the reference input has to be calculated. For the determination of the next reference input the following two cases can occur:

Use of reference from the setpoint changer: The setpoint changer calculates the reference from the two newest setpoints requested from the user. If there is no difference between the two setpoints it will set the reference to the newest setpoint, or else it will increase the reference according to the fastest preset ramp.

Use of reference from the user defined ramp: If the user has requested a ramp, the fastest ramp is followed to the start point of the user defined ramp. Afterwards the reference will increase (or decrease) to follow the user defined ramp to the end point.

Hereby the values needed for the control signal determination algorithm are obtained.

10.3.4 Calculate the Next Control Signal

In chapter 8 it was chosen to use a PID controller. And the following difference equation was obtained:

$$\begin{aligned} u[n] = & \\ & e[n](K + KT_d) + e[n-1]\left(K\left(\frac{T_s}{T_i} - 1 - 2T_d\right)\right) \\ & + e[n-2]KT_d + u[n-1] \end{aligned} \tag{10.4}$$

The flow chart for the control algorithm is shown in figure 10.6. The constant MAX_OUTPUT is defined as 10 which is the maximum voltage output for the data acquisition card.

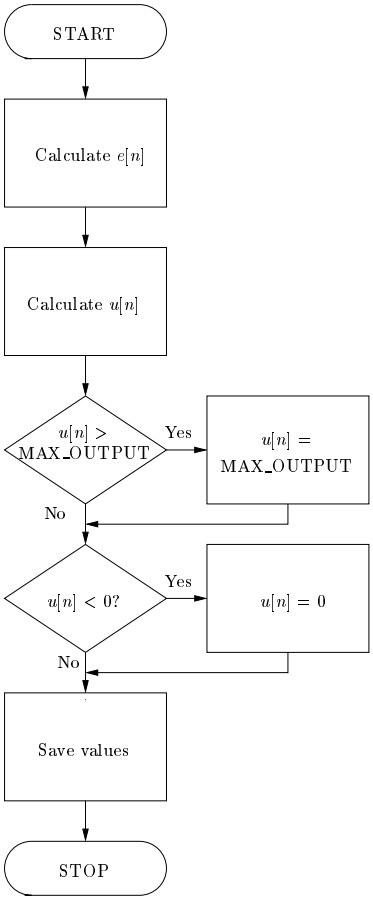


Figure 10.6: Flow chart for the control algorithm.

When the next control signal has been determined it is sent to the data acquisition card by using the function `AO_VWriteChannel` function written in the device driver for the data acquisition card. This function needs a parameter for the output channel number where the voltage has to be written and the voltage to be written.

When the output has been written, the rest of the time, before next sample time, is used to send data to and receive data from the GUI process.

10.3.5 Send Data to the GUI Process

In the description of the protocol in section 10.1 the data package to be sent to the GUI process is defined.

Before the data package can be send, the actual air velocity, temperature and ramp status needs to be converted to characters as the protocol describes. When this conversion has been done, an array of 14 bytes is written to the pipe.

10.3.6 Receive Data from the GUI Process

When the data is send, the controller has to receive data from the GUI process. The data packages to be received from the GUI process are described in the protocol in section 10.1.

The receive data algorithm is shown in the flow chart of figure 10.7.

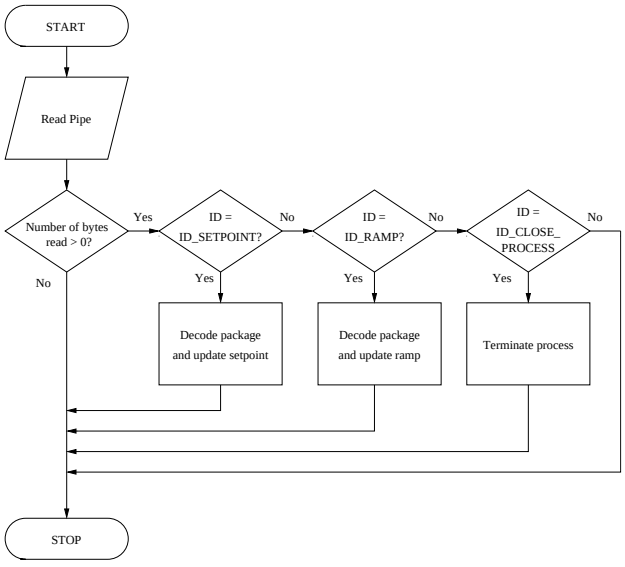


Figure 10.7: The algorithm to receive data from the GUI process.

In the flow chart it can be seen that the receive data algorithm simply reads

from the pipe, and determines the identifier of the package received and acts in accordance to the indentifier.

When the control loop is finished the controller process goes into a wait loop until the `CreateTimerQueueTimer` generates a new interrupt.

10.4 Software Test

In the accept test specification the following demands were given to the GUI (see section 3.12):

Demand 1: All text in the GUI must be in English.

Demand 2: All buttons and text fields must be accessible with both mouse and keyboard and functional.

Demand 3: The numbers typed in the text fields must be of the format xx.x, where the x'es are the numbers from 0 to 9 and the "." is a decimal delimiter. The numbers after the "." are optional. The range of input air velocity is from 0 to 30 $\frac{\text{m}}{\text{s}}$. If an invalid input is given, the GUI must display an error message.

Demand 4: The GUI must give the user an indication on whether the wind tunnel is "ready" or "not ready" for measurements. This must be done by displaying a red or green indicator respectively.

Demand 5: The software must produce an ASCII-text file in .csv format, containing logged information about the air velocity and temperature from the last invocation of the "Submit" button. The log file will be stored on the harddisk of the PC. A MATLAB™ program `PelecanPlot` can be used to load the data from the .csv file and to make a plot of the measurements.

For each of these demands a test is defined. In the following each of these tests are carried out to verify that each demand is fulfilled by the software.

10.4.1 Test 1

To this demand the following test is specified:

Test 1: Start up the GUI and navigate through all the menus, buttons and pop up windows, verifying that they are in English.

By running this test it can be verified that the text in the GUI is in English by reading the text on the screen.

10.4.2 Test 2

For this demand the following test is specified:

Test 2: Start up the GUI and navigate through the screen with only the mouse. Ensure that all areas can be reached. Repeat the test only using the keyboard. Ensure that all areas can be reached.

The keyboard and mouse was used to navigate through the menus, buttons and pop-up windows and it was verified that only the mouse can be used to handle all the options.

10.4.3 Test 3

For this demand the following test is specified:

Test 3: Start the GUI. Move to a text field and type a legal number and see that the software accepts the value. Type a value that is out of range and see that the system gives an error and rejects the input. Type in strings that contain illegal symbols or are of an invalid format, and see that the system displays an error message and rejects the input.

The text field for setting a user defined air velocity was selected and four different legal air velocity was inserted in the text field: one with the form x, one with the form xx, one with the form x.x, and last one with the form xx.x. These air velocity values was all accepted by the GUI.

The maximal air velocity that can be entered is in the demand specfied to 30 $\frac{\text{m}}{\text{s}}$ but because of limitations in the pressure transducers the maximal air velocity that can be entered i the GUI is 24 $\frac{\text{m}}{\text{s}}$.

Afterwards four illegal air velocity values were inserted: a character, numbers of the form x.xx, a number higher than 24 $\frac{\text{m}}{\text{s}}$ and last nothing was inserted. For each of these illegal inputs an appropriate error message was displayed.

This test was then repeated for the ramp `From` and ramp `To` text fields. The same results were verified with these two text fields.

Then the ramp `Time` text field was tested. First, legal values were typed in the test field: integers in the range of 0 to 1440, decimal numbers in the range of 0 to 1440 with a maximum of three decimals. 1440 min equals the demand of 24 hours. All these values were accepted, if it was ensured that the ramp slope was within the given range. Otherwise an error message was displayed. Finally, illegal values were inserted in the text field. When the `Submit` button was pressed, appropriate error messages appeared on the screen.

10.4.4 Test 4

For this demand the following test is specified:

Test 4: Start the GUI and type in a desired air velocity in the appropriate text field. Submit the value and verify that the air velocity in the wind tunnel starts building up. As the desired air velocity is submitted the indication "not ready" will be shown and when the air velocity reaches the desired value the indication will switch to "ready".

An air velocity was inserted in the text field for the user defined air velocity and the `Submit` button was pressed. By reading the text field with the actual air velocity in the wind tunnel it could be verified that the red indicator was active while the air velocity in the tunnel was out of the range of an acceptable air velocity; that is outside a range of ± 1 % of the submitted air velocity. When the air velocity came within this range the green indicator became active.

10.4.5 Test 5

For this demand the following test is specified:

Test 5: Start the GUI and submit a legal air velocity in the text field. Let the air velocity build up and press the stop button after an appropriate time. Go to the file menu and choose the save option. Enter a file name and a location for the file and press "save". Open a MATLAB™ command window and choose the "import" option in the file menu. Find the newly saved file from the wind tunnel control program and ensure that it has the ".csv" extension. Load it into MATLAB™ with `PelecanPlot` and see that the output from `PelecanPlot` contains information of the time span, air velocity and temperature of the just performed test in three separate vectors.

The GUI was started and an air velocity was inserted in the text field for the user defined air velocity and the `Submit` button was pressed. After the submitted air velocity was reached the `STOP` button was pressed, and the air velocity in the wind tunnel graduately changed to 0 $\frac{\text{m}}{\text{s}}$.

The file menu `save` was selected and the logged data was saved to the disk. A MATLAB™ command window was opened and the data was loaded with the `PelecanPlot` function and the logged data where drawn on the screen. By inspection of the logged data, it could be verified that the structure of the file was correct.

10.5 Conclusion

The purpose of the software is to give a user friendly interface to the wind tunnel control. It is written to give the user a possible way to interact with the wind tunnel: see the actual air velocity and temperature in the wind tunnel, set the air velocity, make the air velocity follow a defined ramp and save logged data to a .csv file.

As verified in the test, the software has these possibilities except that the GUI can not be controlled by the keyboard alone.

Chapter 11

Accept Test

In the accept test specification in section 3.12 a set of demands was given to the system. The first 5 demands are associated with the GUI and have been tested in section 10.4. The remaining demands are related to the entire system and are repeated here:

11.1 Demands for the Control Algorithm

Demand 6: The sample frequency must ensure that the shortest rise/fall time is sampled at least 10 times.

Demand 7: The controller must not be so fast that the VLT ramp limits the speed. The accuracy of the controller must be $\pm 1\%$, that is a maximum of 0.3 $\frac{\text{m}}{\text{s}}$ at 30 $\frac{\text{m}}{\text{s}}$. A 0.3 $\frac{\text{m}}{\text{s}}$ step, corresponding to a 100 mV input step, takes 10 ms when the ramp time from 0 V to 10 V is set to 1 s. It means that the rise time and fall time of the controller must be at least $t_r > 0.08$ s.

Demand 8: The overshoot must be within the demand of $\pm 1\%$ accuracy.

Demand 9: The system must have no tracking error for ramp input.

Demand 10: The system must deliver a stable air velocity, within $\pm 1\%$ of the desired value, in the wind tunnel within the limits of 0–30 $\frac{\text{m}}{\text{s}}$.

11.1.1 Test 6

For this demand the following test is specified.

Test 6: Use the procedure of test 5 to save and load a file into MATLAB™ and use the `plot` command to plot the air velocity response vs. the time vector and compute the sample time. Read the rise time from the plot and compute the number of samples on a rise time by multiplying with the sample frequency. According to the demand this has to be at least 10.

On figure 11.1 a plot of the step response from 15 $\frac{\text{m}}{\text{s}}$ to 17.5 $\frac{\text{m}}{\text{s}}$ is shown. The rise time is read to be 1.8 s and hence the number of samples on a rise time is 90. This controller fulfills the given demand.

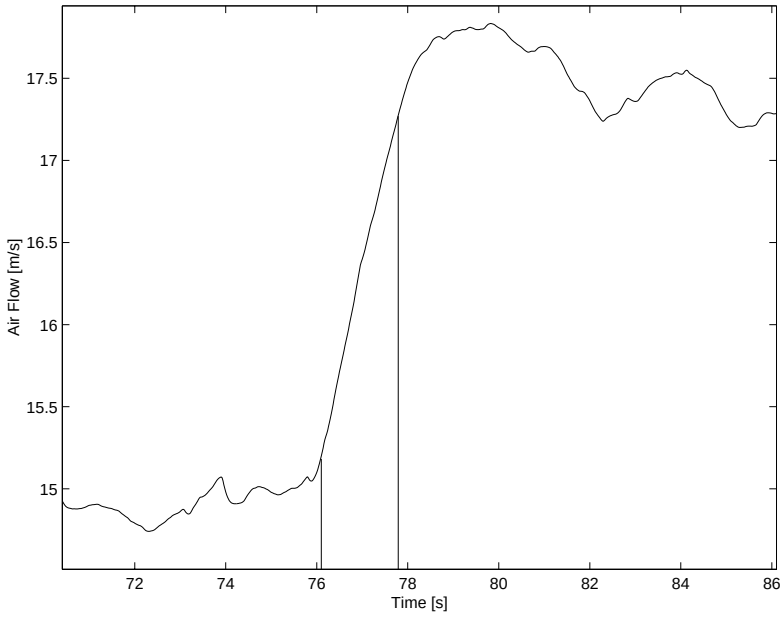


Figure 11.1: Rise time for the system with the PID controller.

11.1.2 Test 7 and 8

To these demands the following test is specified.

Test 7 and 8: Use the procedure in test 5 to produce, save and load a positive step response into MATLAB™. Plot the response vs. time with the `plot` command. Verify that the overshoot present is within 0.1% of the steady state air velocity. Compare the submitted value with the last recorded values of the test, and see that they are the same within ±1%. Compute the rise time and verify that it fulfills the demand. Repeat the procedure for a negative step.

The error in steady state is read from a plot in MATLAB™ to be approximately ±2% by which the demand is not fulfilled. This error could be caused by noise from the sensors or from the control signal. This could probably be reduced by introducing a better filter than the one used in the PID controller with filter in section 6.4 on page 47.

A way to further improve the controller is to introduce an inner feedback loop to control the speed of the AC-motor. This was not done as it was not possible to install an encoder on the motor.

The overshoot shown on figure 11.1 is approximately 20 % which is greater than the demand allows. This, however, is not critical to the operation of the system. To eliminate the overshoot, a detuning of the controller could be made, which would also make the system respond slower.

11.1.3 Test 9

To this demand the following test is specified.

Test 9: Specify a ramp input to the system and submit the input. Use the procedure of test 5 to save and load the result into MATLAB^{TM}. View the resulting output with the `plot` command and compare it with the submitted ramp. The system response must follow the specified ramp within $\pm 1\%$.

To illustrate the ramp error a plot of a ramp input from 0–20 $\frac{\mathrm{m}}{\mathrm{s}}$ in 2 min is shown on figure 11.2.

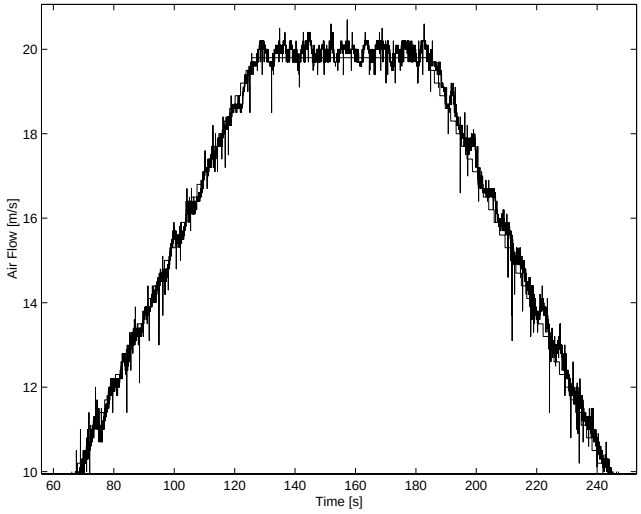


Figure 11.2: Response to a ramp input plot together with the corresponding input ramp.

This ramp input is chosen because a steeper ramp is never used for measurements and a flatter ramp will make the ramp error smaller. The ramp error for the chosen ramp is approximately zero, which fulfills the given demand.

11.1.4 Test 10

To this demand the following test is specified.

Test 10: Submit different values of air velocity for the system and let the same velocity be present for 3 min. Examine the resulting air velocity with Matlab^{TM} and with a separate hand held air velocity meter in the test section. The air velocity must not vary more than the specified $\pm 1\%$.

It was not possible to make this test due to the calibration errors of the pressure sensors.

Chapter 12

Conclusion

The purpose of this project was to design and implement a controller for a wind tunnel. The tunnel is owned by the Institute of Energy Technology and is primarily used for scientific purposes. Therefore the project has been made in cooperation with the Institute of Energy Technology.

The control of the wind tunnel should be accessed through a graphical user interface (GUI) with the possibility to give readouts of the current temperature and air velocity in the test section of the wind tunnel and to let the user specify the progress of the air velocity in the test section. Moreover, the air velocity and temperature in the test section of the tunnel should be stored in a log file.

To be able to design the controllers, the wind tunnel was analyzed and a mathematical model was derived. In addition to that, an experimental model of the total wind tunnel system, including the motor and VLT, was made. The experimental model was used since it was not possible to derive a usable mathematical model for the motor and VLT.

It was the hope that a better model of the wind tunnel could be obtained before the end of the project, but this was not posible since the real measuring equipment was not installed before the end of the project. The consequence of this is that the experimental model is based on measurements with a handheld instrument and the controllers have been designed on this basis but implemented and tested in the tunnel with the right sensors.

The demands for the control system and for the controllers were structured in the demand specification and six modules were specified in the modulization. The modules are: Setpoint Changer, Classical Controller, State-space Controller, Choice of Controller, Hardware Configuration and Software. After this, three different controllers have been designed and tested in the wind tunnel, namely: a PID controller, a PID controller with filter and a state-space controller. This concludes that the project covers both modern and classical control strategies.

Each of the three controller designs follow the strategy of making a continuous-time prototype followed by a discretisation to enable the implementation on a PC. This approach meant that the PID controllers had to be adjusted after

the design in order to fulfill the discrete-time demands. This could be avoided by using a discrete-time design method instead.

Due to the very late instrumentation of the wind tunnel and a miscalibration of the pressure transducers, the measured and tested controllers did not conform to the calculated and simulated ones.

After a tuning of the PID controller it was, however, possible to successfully apply this on the wind tunnel whereas the state-space controller and the PID controller with filter did not prove to be suitable solutions with the present models.

The simulated PID controller with input filter could be implemented directly in the wind tunnel without any further adjustments, but the rise time of the tested closed-loop system was measured to be 4 seconds instead of the simulated 0.7–0.8 seconds.

The implemented GUI works and fulfills the demands. It is posible to control the tunnel through the GUI and it offers the opportunity to store the air velocity and temperature in a log file. The data in the log file can subsequently be viewed in MATLAB^{TM} e.g. and used for further analysis.

12.1 Improvements

The performance of the controllers can be improved with a better tunnel model after a recalibration of the sensors. On top of that, the controllers could be designed to have a zero tracking error.

The temperature readouts on the GUI are clearly influenced by electrical noise. This can be improved by placing the measuring circuit (the Wheatstone bridge and amplifier) near the PT100 sensor and transmitting the 0–10 V signal to the data acquisition card. The PCB layout for this measuring circuit is enclosed in appendix J.

To further reduce the influence of noise on the controller, the input channels on the data acquisition card could be equipped with active lowpass filters with a 0 dB DC-gain.

To make the air velocity readouts more precise, the temperature must be included in the calculation of the air velocity.

An improvement of the GUI would be the opportunity to specify the progress of the air velocity by defining the velocity at different times as sketched in figur 12.1.

In total the project has produced a basis on which a fully functional control system can be built after a sensor calibration and thereby a better model of the

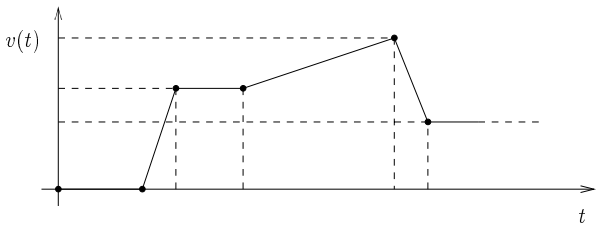


Figure 12.1: Specification of times and velocities in the air velocity.

system. The ideas pointed out through this project can be used as guidelines to improve the wind tunnel control system.

Bibliography

[cdrom] Enclosed CDROM for this project

[dc] Franklin, Powell & Workman *Digtal Control of Feedback System* Addison Wesley ISBN: 0201820544

[fc] Franklin, Powell & Emami-Naeini *Feedback Control of Dynamic System* Addison Wesley ISBN: 0201527472

[haugen] Finn Haugen *Regulering av dynamiske systemer* Tapir ISBN: 82-519-1433-7

[ogata] Ogata, Katsuhiko *Modern Control Engineering, second edition* Prentice-Hall, Inc., 1990 ISBN: 0135987318

[pukkila] Pukkila, Olli *Handbook of Modern Fan Technology* Ilmateollisuus, 1987

[ståbi] *Ventilation ståbi* Teknisk Forlag A/S, 1990ISBN: 8757111073

Appendix A

Simulations

In this appendix the simulations of the three controllers will be described.

The conclusions to the simulation of each controller is placed in their respective chapters.

A.1 Simulation of Classical Controllers

In order to validate the different controller designs further, a simulation is used. The simulation is done using the MATLAB™ toolbox SIMULINK™. The things to be examined through the simulations are:

The use of control signal: This will be measured directly in the simulation model.

The noise sensitivity: This is done by introducing noise on the reference and on the control signal. The sources will be either sine or random signals.

The response to ramp input: This will show the effects of the steady state errors to ramp input.

A.1.1 The Simulation Model

The model contains a discrete-time realisation of the controller and a continuous-time representation of the system. This is done since this will be the real configuration of the system when it is implemented in the tunnel. This also means that the control signal can not be altered faster than the sample frequency allows. Further, a 0–10 V saturation block is inserted in the model to limit the use of control signal. The limits are set such that the low limit is 0 V and the high limit is 10 V. This is done to simulate the limitations in the output signal introduced by the limit of output voltage in the data acquisition card. The model is shown in figure A.1.

The same model is used to simulate the controller with input filter, the only difference is that the discrete-time realization of the filter is placed in front of

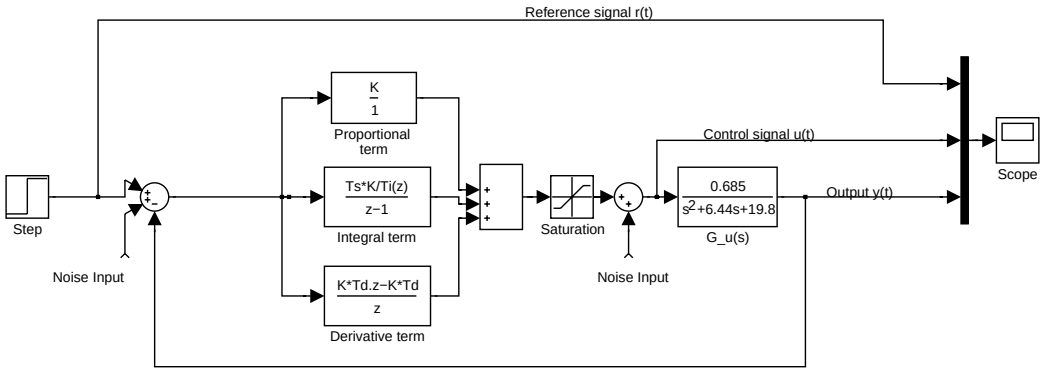


Figure A.1: The Simulink™ simulation model of the PID controller without input filter.

the controller.

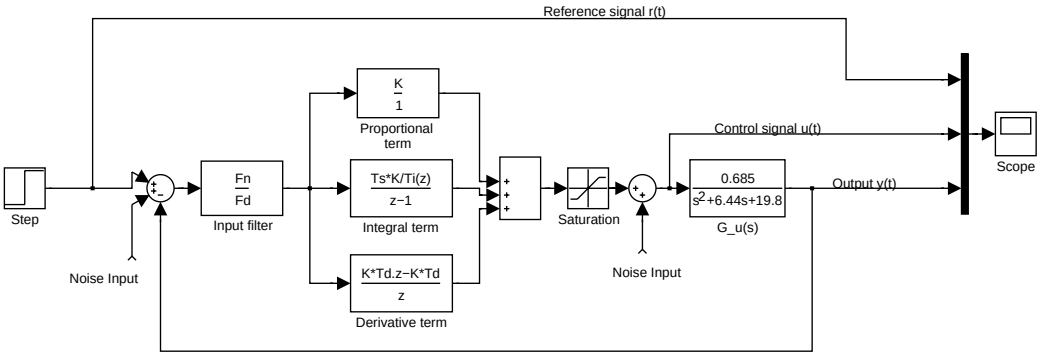


Figure A.2: The Simulink™ simulation model of the PID controller with input filter.

The filter and the controller parameters are not shown directly in the model. Instead they are given by names and vectors. This is to avoid rounding error that can arrise from typing the numbers directly in the model. The parameters are therefore passed directly from the Matlab™ workspace to the simulation model through the different variables.

Since the models includes the transfer function $G_u(s)$ the inputs needs to be scaled to maximum 0.35. If it is desired to use input signals from 0–10 V, an additional gain must be introduced in the feedback path, and just before the controller.

A.1.2 Simulation Results

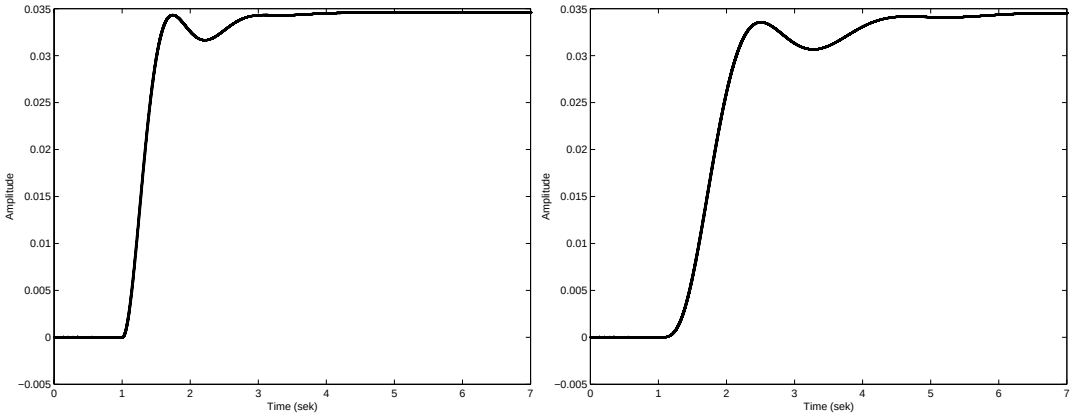
Noise Rejection

In this part of the simulation the controller's ability to reject noise is examined. The noise is introduced at the places where analogue signals are present in the real setup, that is at the control signal and the messured signals (see figure A.1 and A.2).

The simulation is run with two types of noise input: first with a sine input with frequencies 5 Hz, 50 Hz and 1 kHz and then random noise with a variance of 0.0692. The choice of frequencies for the sine input are meant to cover the different noises from the slow control signal, the source voltage and the switch harmonics. The noise is chosen to have an amplitude twice the amplitude of the reference signal. The reference input is chosen to be a step from 0 to 0.0346 as in the previous simulation.

To be able to determine the noise rejection ability of the controller, the noise is first introduced in the control signal, then in the feedback path. This is done with all three noise frequencies and the random noise. The results are displayed on the scope and in figures A.3 to A.5

Sine Disturbances on the Control Signal

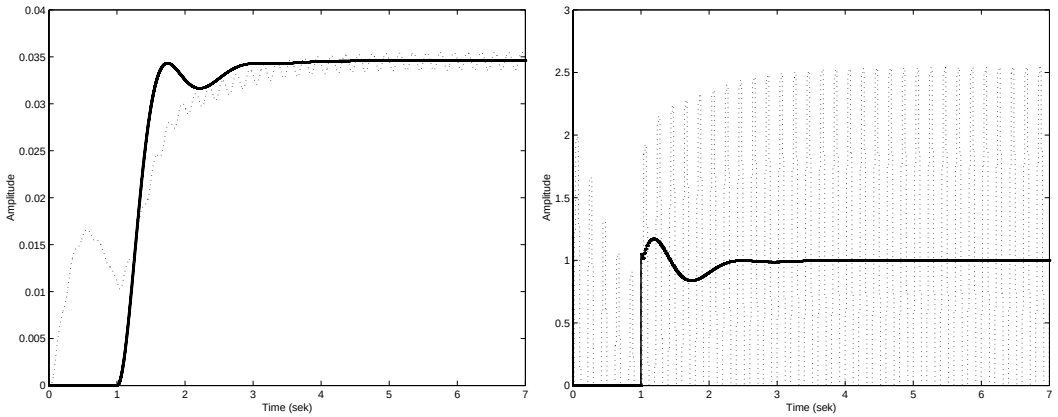


(a) (b)

Figure A.3: (a) The output of the controller without filter with sine noise on u. (b) The output of the controller with filter and sine noise on u.

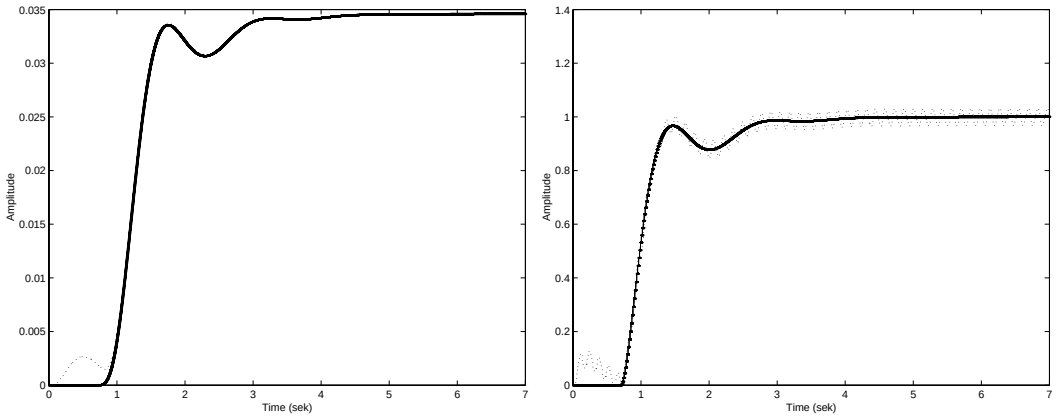
Sine Disturbances on the Feedback Path

In the following the solid line is the output signal without disturbance the dottet, dot-dash and dashed line are the output signal with 5 Hz, 50 Hz and 1 kHz noise respectivly.



(a) (b)

Figure A.4: (a) The output signal of the closed-loop system with no filter and sine noise in the feedback. (b) The control signal of the controller without filter with sine noise in the feedback.

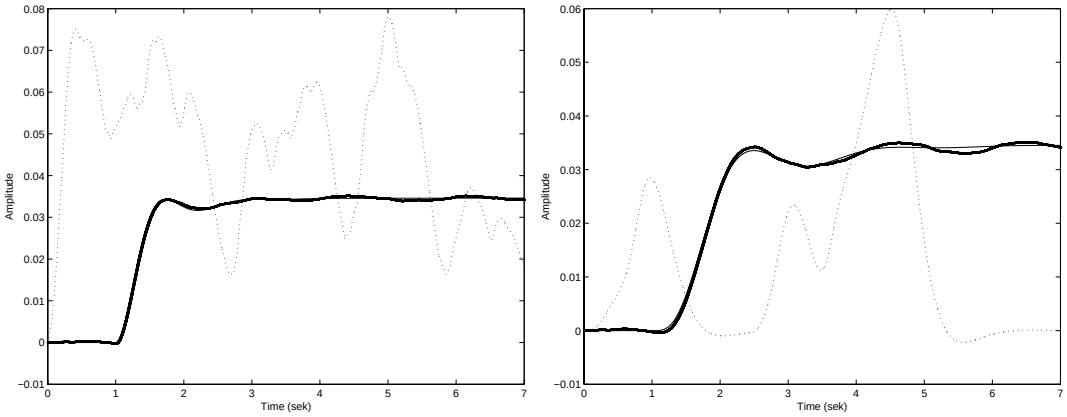


(a) (b)

Figure A.5: (a) The output of the controller with filter and sine noise in the feedback. (b) The control signal of the controller with filter and sine noise in the feedback

Random Noise Disturbance

In the following parts of the simulation the solid line indicates no disturbance, the dotted line is noise on the feedback path, the dot-dash line is noise on the control signal and the dashed line is noise both places.



(a) (b)

Figure A.6: (a) The output of the controller without filter with random noise in the Feedback. (b) The output of the controller with filter and random noise in the feedback.

Ramp Response

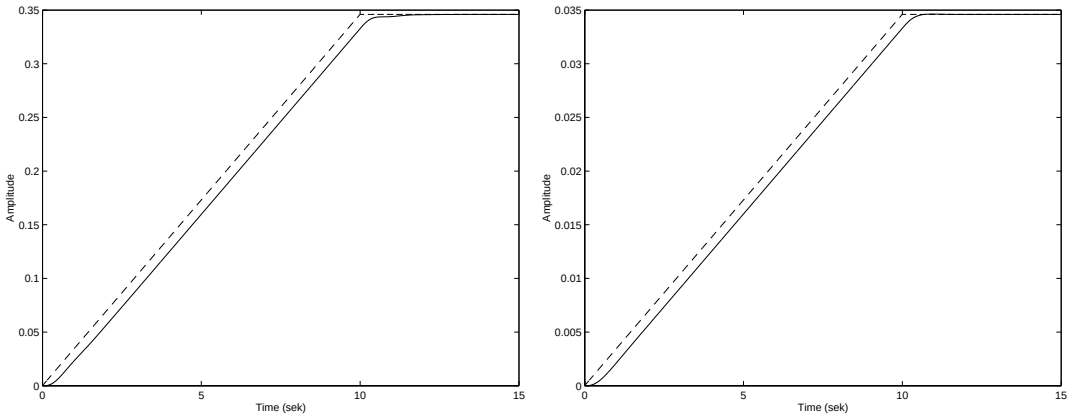
The purpose of this simulation is to see how the closed-loop system responds to a ramp input. This means that the effect of the tracking error for ramp input will be determined by simulation. The steepest ramp allowed is the ramp specified by the setpoint changer. This means a ramp with an incline of 0.0346, that is a ramp time of 10 s from zero to 100% output. This ramp is used together with a ramp with an incline of 0.00346 to prove that a higher incline, results in a larger tracking error.

The ramp in the simulation is produced by the ramp source and a saturation block to terminate the ramp. The results of the simulation are displayed in figure A.7 and A.8.

The response to ramp input shows that the tracking error is reduced with the ramp incline as expected.

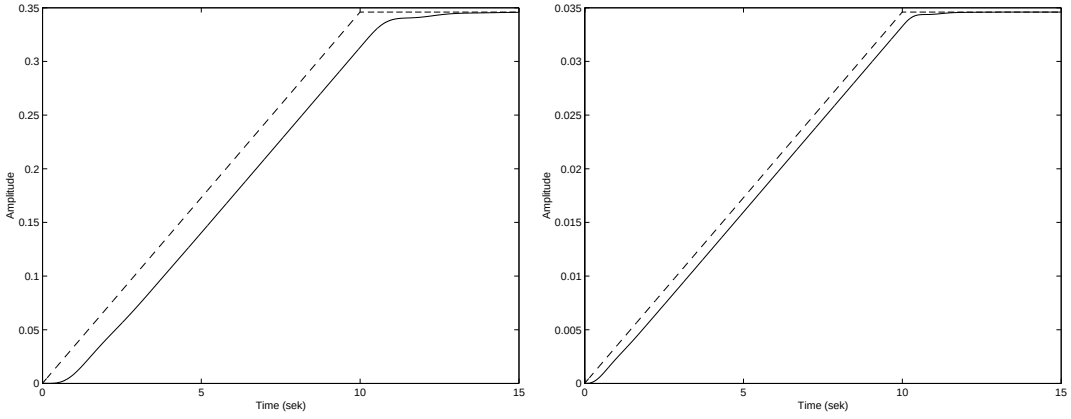
Control Signal

The purpose of this simulation is to examine how much control signal the controller uses to control the plant. This has to be investigated since the control



(a) (b)

Figure A.7: The response of the system with no filter to the fastest ramp input (a). The response of the system with no filter to the slower ramp input (b).



(a) (b)

Figure A.8: The response of the system with input filter to the fastest ramp input (a). The response of the system with input filter to the slower ramp input (b).

signal can only be between 0 and 10 V. This also means that the controller will not work optimal at setpoints close to these limits. The simulation is performed by setting the input step to 0.0346, which corresponds to a 1 V reference step. The step is set to start after one second. The resulting changes in the control signal can then be seen on the scope and in figure A.9.

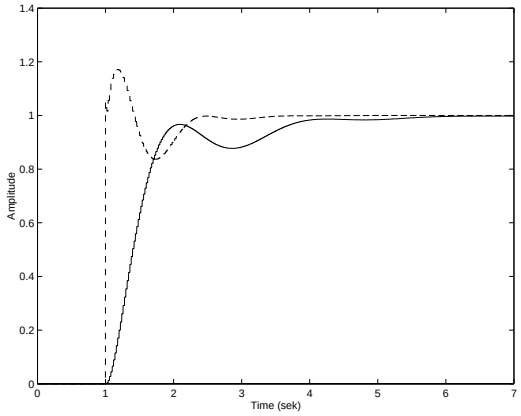


Figure A.9: The use of control signal for the two controllers.

In general the controller with input filter uses less control effort, i.e. control signal, in any of the simulations than the controller without filter. This is because the filter removes many of the disturbansis that the controller must take care of in the controller without filter.

A.2 Simulation of the State-space Compensator

Below is a list of the simulations of the state-space compensator, simulated in MATLAB™ and SIMULINK™:

- A step response without any disturbances.
- A step response with a 2 V sine disturbance on u.
- A step response with a sine disturbance on the feedback path from y with an amplitude of $\frac{1}{10}$ of the signal amplitude.
- A step response with a random signal on u.
- A step response with a random signal on the feedback path from y.
- A ramp response without any disturbances.

All step responses have been made with a step reference input from 0 to 0.0346, resulting in a step from 0 to 1 V on the control signal u. The frequencies of the sine signal is 5 Hz, 50 Hz and 1 kHz as in the simulation of the PID controller.

A.2.1 The Simulation Model

The simulation model is shown in figure A.10. It has the same features as the

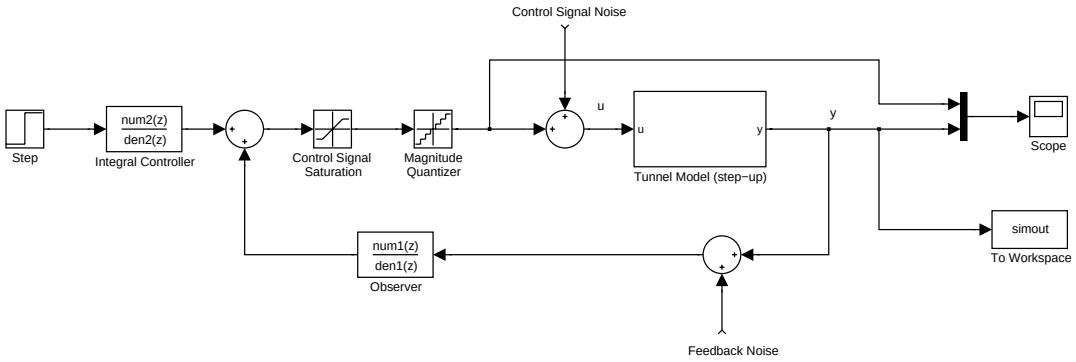


Figure A.10: Simulink™ block diagram of the simulation model.

block diagram in figure 7.5 but with the following additions and modifications:

- The compensator has been replaced with the two discrete-time transfer functions of equations 7.40 and 7.41.
- A saturation element has been added on the output of the compensator with a maximum limit of 10 V so that the control signal u cannot exceed this value.
- A magnitude quantizer has been added after the output of the integral controller. The magnitude resolution is $\frac{10}{2^{11}}\ \frac{\mathrm{V}}{\mathrm{digit}}$ in accordance with the I/O card specifications.

The data of the step responses have been stored in a variable `simout` and have subsequently been plotted in Matlab™. Only the plots from the step-up simulations have been included.

A.2.2 Simulation Results

No Disturbances

Figure A.11 shows the step responses of the discrete-time system (solid line) and the continuous-time system (dashed line) with no disturbances added.

From the figure it can be seen that the discrete-time tranfer function has a different shape than the continuous-time tranfer function. Increasing the sample frequency, i.e. decreasing `Ts`, will move the discrete-time response towards the continuous-time response. Apart from the shapes of the responses, the features (i.e. the rise time, settling time, etc.) are similar.

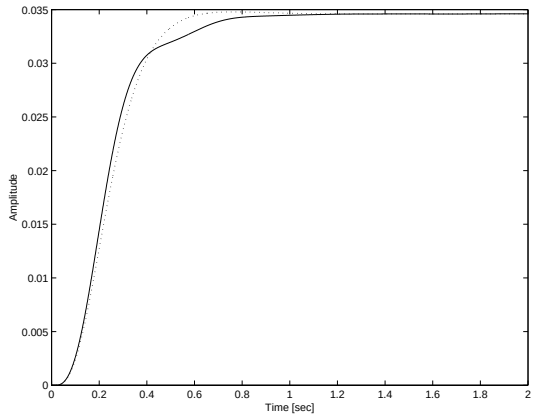


Figure A.11: Step responses with no disturbances. The solid line is the discrete-time system and the dotted line is the continuous-time system.

Sine Disturbances on the Control Input

Figure A.12 shows the step response without disturbances (the solid line) and with three sine signals added. The amplitude of the sine signals are 2 V. The

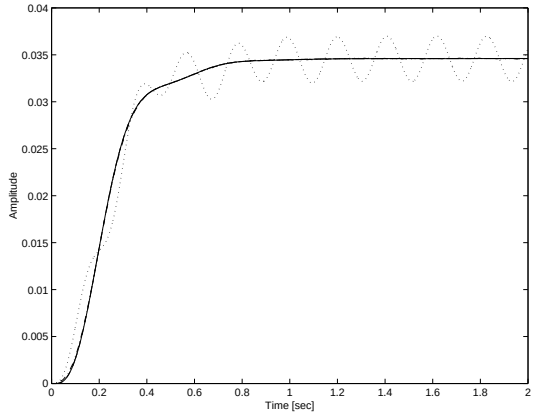


Figure A.12: Step response with sine disturbances on u.

dotted line shows the response of a 5 Hz signal and it clearly reflects the sine signal on the normal step response. This is related to the bandwidth ω_b of the closed-loop system: Systems with a large bandwidth have shorter rise times and settling times and are therefore more sensitive to fluctuations in the control and reference signals. Systems with a small bandwidth are slower but less sensitive to noise. The bandwidth of the system is approximately $\frac{1.8}{t_r} = 5.8\ \frac{\text{rad}}{\text{s}}$. This formula only gives a rough estimate since it applies to second order systems.

The dot-dash line and the dashed line are the responses of a 50 Hz and a 1 kHz signal respectively. These frequencies clearly have very little effect on the system.

Sine Disturbances on the Feedback Path

Figure A.13 shows the step response without disturbances (the solid line) and with three sine signals added. The amplitudes of the sine signals are $\frac{1}{10}$ of the

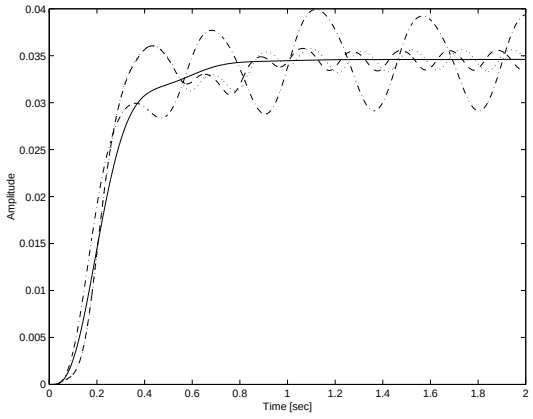


Figure A.13: Step response with sine disturbances on the feedback path.

feedback signal, i.e. 0.00346. The dotted line is the response of a 5 Hz signal. The dot-dash line and the dashed line are the responses of a 50 Hz and a 1 kHz signal respectively. The 5 Hz and the 1 kHz signal produce approximately the same error, whereas the 50 Hz signal produces the biggest error.

Random Signal Disturbances on the Control Input

Figure A.14 shows the step response without disturbances (the solid line) and with two random signals added to the control input. The dotted line is the

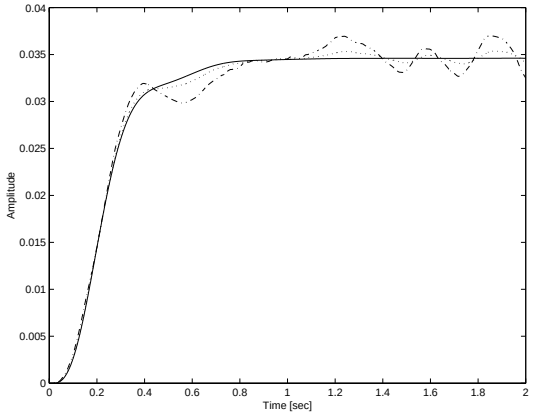


Figure A.14: Step response with random noise on u.

response of a signal with an amplitude variance of twice the control signal (peak values of approximately ± 5 V) and has very little influence on the output.

The dot-dash line is the response of a signal with an amplitude variance of 20

times the control signal (peak values of approximately ±17 V) and this has some influence on the output.

Random Signal Disturbances on the Feedback Path

Figure A.15 shows the step response without disturbances (the solid line) and with two random signals added on the feedback path. The dotted line is the

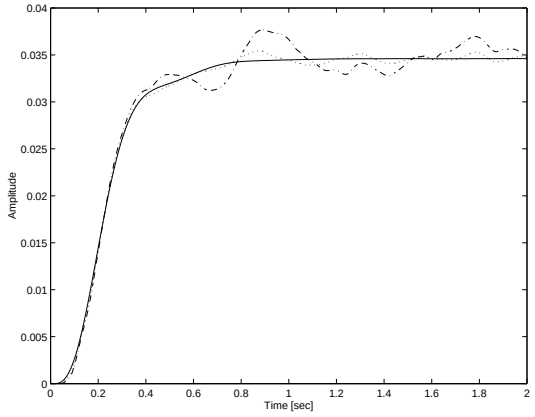


Figure A.15: Step response with random noise on the feedback path.

response of a signal with an amplitude variance of $\frac{1}{100{,}000}$ of the feedback signal (peak values of approximately $\pm\frac{1}{15}$ of the feedback signal) and has only little influence on the output. The dot-dash line is the response of a signal with an amplitude variance of $\frac{1}{10{,}000}$ of the feedback signal (peak values of approximately $\pm\frac{1}{5}$ of the feedback signal) and has a big influence on the output.

Ramp Response

Figure A.16 shows the ramp response without disturbances together with the ramp input.

The slope of the ramp is 0.0346 which gives the maximum input (0.346) after 10 seconds.

From the figure (while plotted in SIMULINK™) it has be read that the tracking error is about 0.0091.

Control Signal

Figure A.17 shows the control signal for a step response without any disturbances (the solid line) and with the 2 V random noise applied to the control input. It can be seen that the control signal has a peak value of approximately 2.5 V for both step responses and that the noise signal results in additional control signal fluctuations with an amplitude of about 0.3 V.

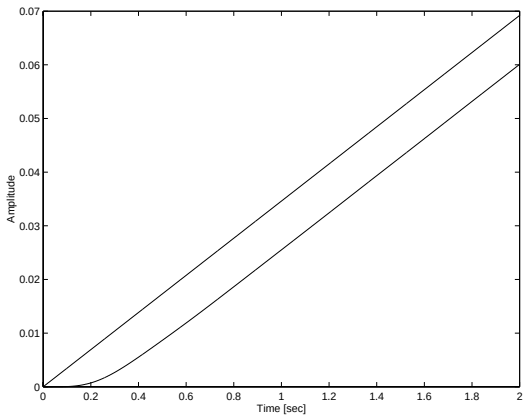


Figure A.16: Ramp response without noise.

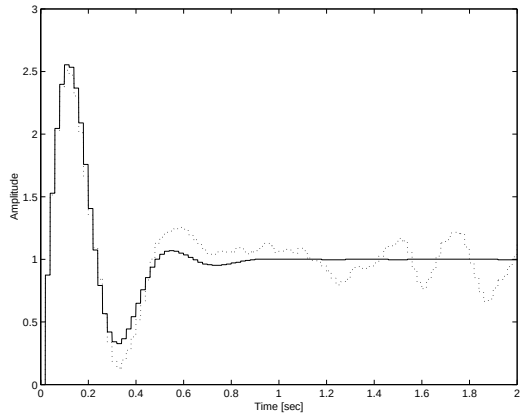


Figure A.17: The control signal for a step response wihtout noise and with random noise on the control input.

Appendix B

Win32 API

To ease the programming of applications for Windows® the developers of Windows® have written functions that can be used in Windows® programming.

The description of, and the header for the C functions that can be accessed through the Win32 API are documented in the Win32api.hlp file [cdrom, win32api.hlp].

B.1 Common Graphical Elements in a Windows® Program

In the Win32 API a number of graphical elements that can be used to create a GUI are defined. Two functions to create graphical elements are defined: `CreateWindow` and `CreateWindowEx`. The difference between these two functions is that with the use of `CreateWindowEx` more options for the window are available. The `CreateWindowEx` is typically used to create the main window of a program because it can also take a user defined window class as an input. In contrast the `CreateWindow` only works with predefined window classes.

B.1.1 User Defined Window Classes

In order to create a user defined window class, a type definition in the Win32 API is available. The type is `WNDCLASSEX` where information about the initial position on the screen, how the icon for the .exe file should look, how the cursor should look in the window, the callback funtion for the window etc. can be set.

When these parameters are properly set the new window class has to be registered as a new window class. When the window class is registered as many window as wanted can be created from this new class using the `CreateWindowEx` function.

B.1.2 Predefined Window Classes

In a window created with the `CreateWindowEx` function more windows can be created either by other user defined window classes or by using some of the predefined window classes. The most commonly used predefined window classes (to be used with `CreateWindow` function) are listed below:

EDITBOX: A box where the user can insert text.

LISTBOX: A box where the program can write text.

BUTTON: A box where the user can click.

STATIC: A box where the program can write static text.

When one of these boxes is to be created with the `CreateWindow` function it needs an identifier. This is done to separate message handling for each of the created boxes. More about message handling will be explained later in this chapter.

B.2 The Windows® Message Queue

The first time a window is created in a Windows® program a message queue is defined. Every time there is any keyboard or mouse activity in a graphical element within the window the Windows® kernel will send a message to the program, where the window was created.

Because it is a common problem to distribute messages from the Windows® kernel to functions that handles the messages (the callback functions) the Win32 API offers a way to do this. If this way is used the last code lines in the `WinMain` function must be as given in program B.1.

Program B.1

```
while(GetMessage(&Msg, NULL, 0, 0) > 0)
  {
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
  }
```

This code will then call the right callback function when it receives a message from the kernel. This message handling is illustrated in figure B.1.

As shown in the figure, 3 parameters are specified in the callback function when a message is sent. The first is the message parameter: this is used by the Windows® kernel whenever there is any activity in a window. Below the most important Windows® messages is listed:

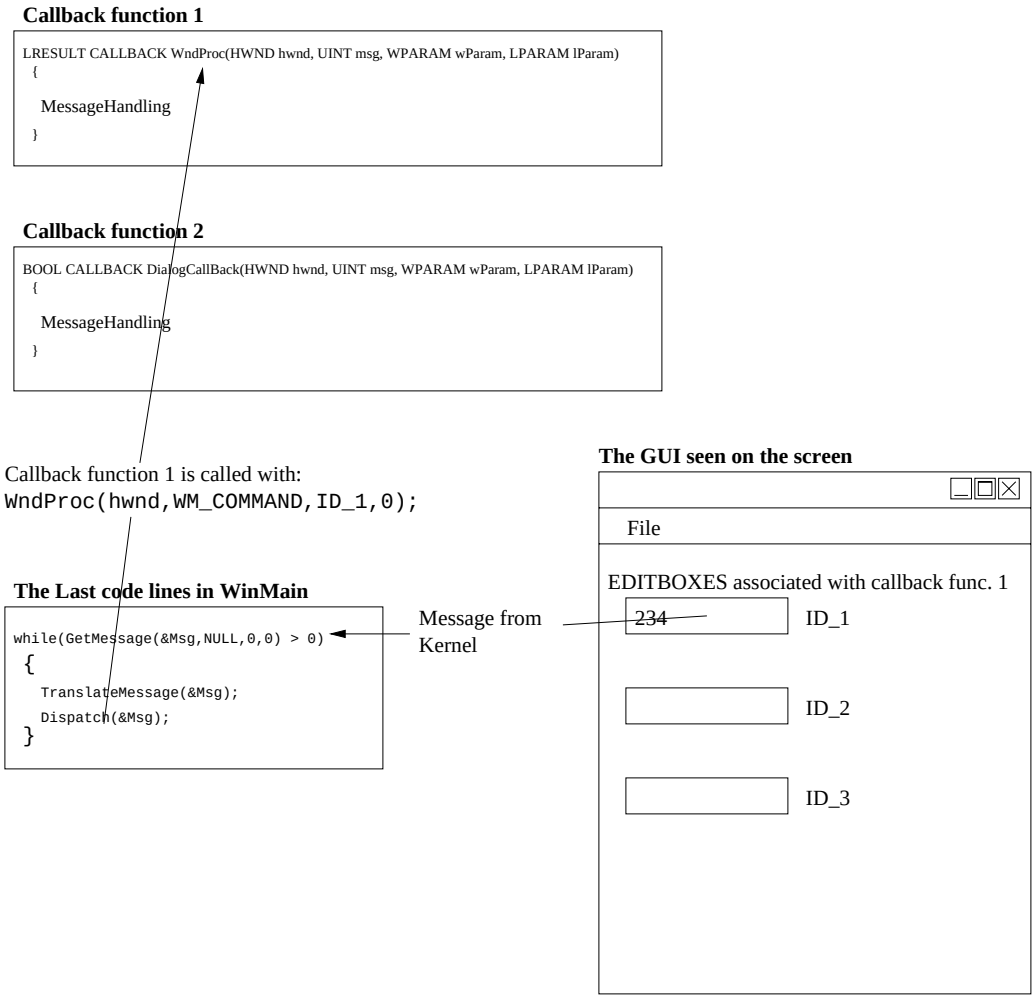


Figure B.1: Illustration of how a user activity will cause a callback function to be called.

WM_CREATE: This message is sent just after the window is created.

WM_COMMAND: This message is sent whenever there is any activity within the specific window.

WM_PAINT: This message is sent when the window needs to be repainted.

WM_CLOSE: This message is sent when the window has to be closed.

WM_DESTROY: This message is sent whenever an application is to be closed or killed.

The two other parameters are user defined. This means that whenever a graphical element is created it needs an identifier. Whenever there is any activity in a graphical element the Windows® kernel will send a message with the message parameter set to WM_COMMAND and the second parameter to the identifier of the element and the last parameter set to 0.

Of course the programmer can send messages to a callback function whenever it is desired. All that is needed to send a message to a callback function is the `HWND` parameter which the Win32 API identifier for windows and dialogboxes. In such a call to a callback function the three other parameters can be specified to whatever wanted.

B.2.1 Message Handling

When the callback functions are called it is quite easy to control the handling of the graphical elements. It is simply a matter of testing the three parameters the callback function is called with and act in accordance with the case that is found.

B.3 Windows® Resource Files

In order to reduce the number of code lines for menus and dialogboxes a Windows® resource file can be written. This resource file has its own compact syntax and therefore it is often recommended to use applications that can write resource files.

When a menu or a dialogbox is created in the resource file it needs an identifier. The messages that is sent when a menu is clicked is actually the same as with all other graphical elements: the message parameter is `WM_COMMAND`, the second is the identifier and the last is 0.

Dialogboxes work exactly like windows. A dialogbox has its own callback function and the elements within it need their own identifiers.

B.4 Pipes

One of the disadvantages of the Windows® message system is that it is slow. Through tests it was seen that messages between two processes cannot be sent with 50 Hz if they are sent with Windows® messages.

A pipe is a way to make a communication between threads or processes (or a tread and a process). It defines some shared memory for in- and output for each end of the pipe. A pipe is a first in first out (FIFO) buffer.

A Windows® pipe actually works like the Windows® file system. When the pipe is created it can be accessed with the normal Win32 API read and write file functions.

A named pipe can be made where the name of the pipe is specified and it is therefore easy to create and connect to. A pipe is client/server oriented. Hence, the server process (or thread) creates the pipe and the client process (or thread) connects to the pipe wi th the same attributes as when the server process created it, e.g. size of in- and output buffers, access attributes (read and write access) etc.

Appendix C

Tunnel Measurements

Test purpose

The purpose of this test is to collect information about the step response of the wind tunnel. The test contains two parts. The first investigates the step response to a staircase input and the second investigates the step response from rest. The first part is used to estimate the transfer function of the system and the second is used to estimate the delay in the system when started from rest. Either part consists of more measurements.

Equipment

- Air velocity and temperature sensor: SwemaAir30.
- PC with test software and I/O-card.
- 1. order analogue low pass filter.

Test description (part one)

The PC is connected as shown in figure C.1.

One of the analogue outputs of the data acquisition card is connected to the VLT analogue input and the output from the air velocity and temperature sensor is connected to an analogue input on the data acquisition card.

The test software produces a series of ten 1 V steps and stores the output in a file. To minimize the influence of noise in the measurements, the input signal to the PC is fed through an analogue low pass filter, just before it enters the data acquisition card. The sample rate is 50 Hz and the filter's 3 dB frequency is 150 Hz.

The air velocity sensor is placed in the middle of the tunnel to obtain the highest air velocities.

The test is performed by executing the test software from the computer, with both an upward and downward going stair. The results of the measurements

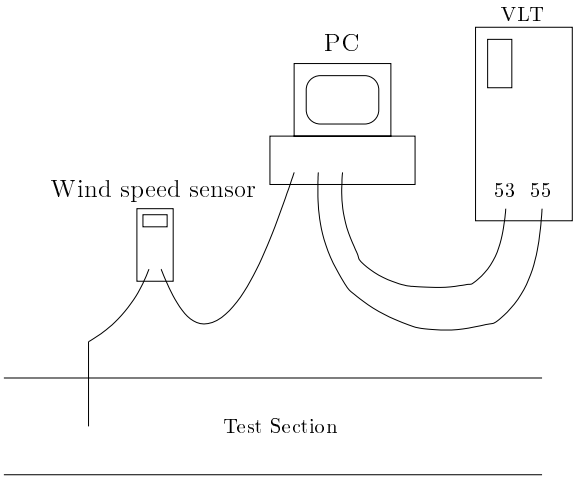


Figure C.1: The test setup for part one of the test.

can be seen on figure C.2.

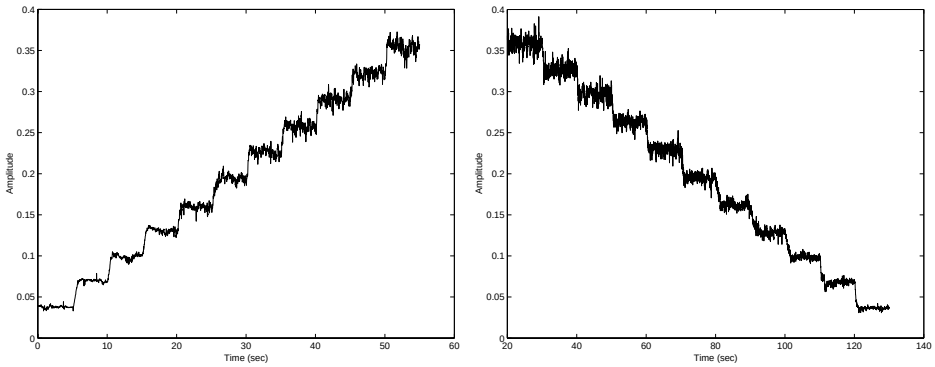


Figure C.2: Step up and step down measurement.

Test description (part two)

The test equipment is connected in the same way as in part one.

The test software produced 4 steps. One from zero to 2.5 V, one from zero to 5 V, one from zero to 7.5 V and the last from zero to 10 V. The responses (figure C.3) were logged with the data acquisition card and stored in a file.

The VLT setup during the measurements is shown in appendix E. The only difference is parameter 207 (Ramp Down Time) which is set to 2 seconds. Otherwise the VLT shows warnings and alarms on the display and sometimes it stops the motor.

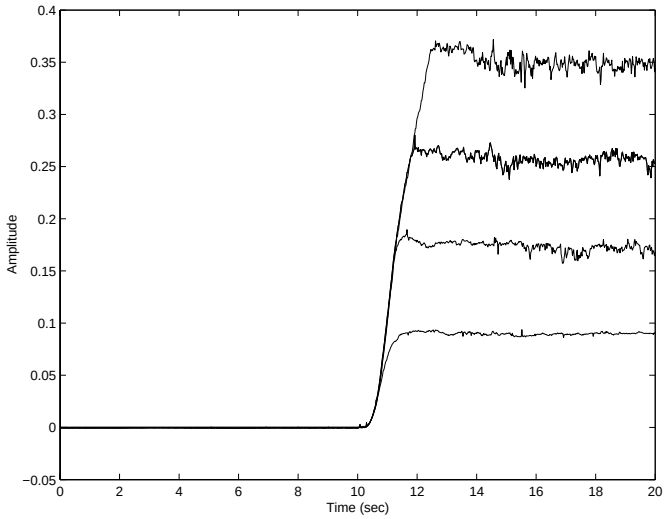


Figure C.3: Four step up measurements (2.5 V, 5 V, 7.5 V and 10 V).

Appendix D

Butterworth Filter

The filter used to filter the measured signals is a second-order Butterworth lowpass filter. The Butterworth filter was used since it has zero ripple in the passband and in the stopband in contrast to a Chebyshev filter e.g.

The filter coefficients are found by trial and error, by plotting the filtered and unfiltered signal in the same window and see if they fit. The plot in figure D.1 was found using the MATLAB^{TM} command `[B A]=butter(2,.04);`.

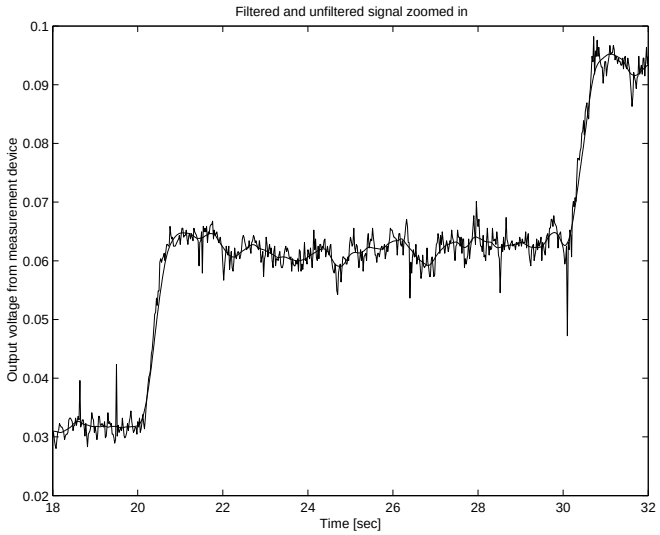


Figure D.1: Plot of the filtered and the unfiltered signal.

Appendix E

VLT Setup

The VLT has to be set up as follows, when the control system is in use. Motor settings and standard settings are not listed here, they are all set in SETUP 1 in parameter "002 Setup" of the VLT. Only important settings for the control system are listed.

Parameter	Setting
002 Active Setup	SETUP 2
100 Configuration	OPEN LOOP
203 Reference Handling	REMOTE
204 Minimum Reference	0.000
205 Maximum Reference	50.000
206 Ramp Up Time	00001
207 Ramp Down Time	00001
210 Reference Type	EKSTERNAL/PRESET
301 Digital Input	FREEZE REFERENCE
302 Digital Input	START
305 Digital Input	PRESET REF.ON
308 Analog Input Voltage	REFERENCE
309 Terminal 53, Minimum Scaling	0.0 V
310 Terminal 53, Maximum Scaling	10.0 V

Table E.1: VLT setup.

Appendix F

Controller test

To verify the controller designs, the controllers have been implemented on a PC and a number of measurements made through the GUI.

The purpose of the measurements is to show:

- The step responses.
- The ramp responses.

To test the controllers, two step responses have been made: one where air velocity steps from 5 $\frac{\text{m}}{\text{s}}$ to 7.5 $\frac{\text{m}}{\text{s}}$ followed by steps from 7.5 $\frac{\text{m}}{\text{s}}$ to 5 $\frac{\text{m}}{\text{s}}$ and one with steps from 15 $\frac{\text{m}}{\text{s}}$ to 17.5 $\frac{\text{m}}{\text{s}}$ followed by steps from 17.5 $\frac{\text{m}}{\text{s}}$ to 15 $\frac{\text{m}}{\text{s}}$.

The steps correspond to a change of 1 V on the control signal, which is the same condition as when the controllers where simulated.

To test how the controllers respond to a ramp input, a ramp test was made with the air velocity graduately increasing from 0 $\frac{\text{m}}{\text{s}}$ to 20 $\frac{\text{m}}{\text{s}}$ and down again. All tests have been made through the GUI where the desired steps and ramps were entered. The data attached to each test was logged in a file and subsequently plotted in MATLAB$^{\text{TM}}$ for inspection. The logged data is the actual air velocity, reference air velocity and the control signal from the controller.

As a consequence of miscalibrations of the sensors it was not possible to take any useful measurements with the sensors. The sensor outputs from a step sequence of ten 1 V steps are shown in figure F.1.

The three low pressure sensors 1, 3 and 5 should be reading 10 V when the air velocity is above 14 $\frac{\text{m}}{\text{s}}$. Refering to the figure, this is certainly not the case and this could imply a calibration error of these.

In order to get acceptable data to cover the air velocity range from 0 $\frac{\text{m}}{\text{s}}$ to 24.8 $\frac{\text{m}}{\text{s}}$ the three high pressure sensors 2, 4 and 6 were used. Therefore, the signal used in the controller is the mean of the values from these three sensors. This value is not entirely correct, as it can be confirmed by regarding the difference between the sensors values on the figure. On the other hand, this is the best possible measurement of the air velocity.

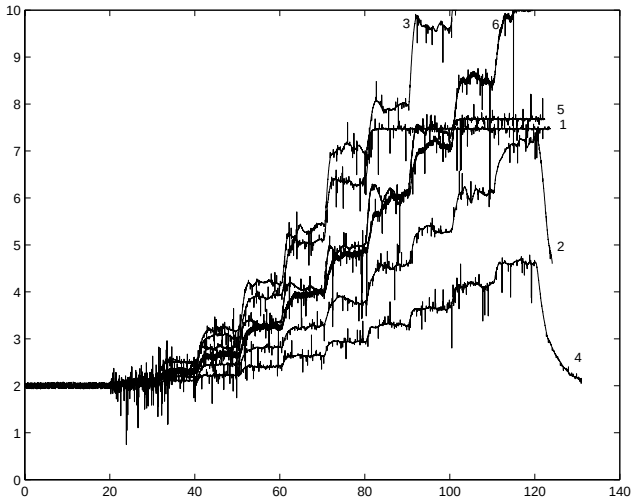


Figure F.1: Output from the six pressure sensors corresponding to input sequence increasing by 1 V in steps.

As a consequence of the installation of new pressure sensors in the wind tunnel, the transfer function for the tunnel has changed after the controllers have been designed.

The new sensor gains are higher than before, resulting in the DC-gain for the tunnel model being a factor 28.9 higher. To compensate for this DC-gain the controllers are adjusted. In the case of the PID controllers it is done by dividing the proportional factor K by the factor 28.9. The state-space controller is changed by dividing the elements of the $\mathbf{C}$ matrix by 28.9, and then finding the new controller equations.

In the following sections the three controller designs are tested with the above mentioned criteria. The steps from 15 $\frac{\mathrm{m}}{\mathrm{s}}$ to 17.5 $\frac{\mathrm{m}}{\mathrm{s}}$ and down are shown together with the ramp responses.

F.1 PID Controller

It was not possible to perform the tests with the PID controller design from section 6.3, because of a current limitation in the VLT. This limitation occurs when the output signal from the controller raises too fast, which means that the controller is to aggressive for the VLT. This error could be a consequence of the new sensors, resulting a change in the tunnel transfer function.

In order to make the PID controller work, a coarse detuning was made to match the dynamics of the tunnel. The tuning was made by running the controller application and observing the wind tunnel while fitting the PID parameters. The new parameters found for the PID are:

$$K = 0.5 \quad T_\mathrm{i} = 0.5 \quad T_\mathrm{d} = 0.05 \tag{F.1}$$

The results from the test are shown in figure F.2 (step input) and F.3 (ramp input), respectively, where parts (a) show the responses from tunnel and parts (b) show the control signals.

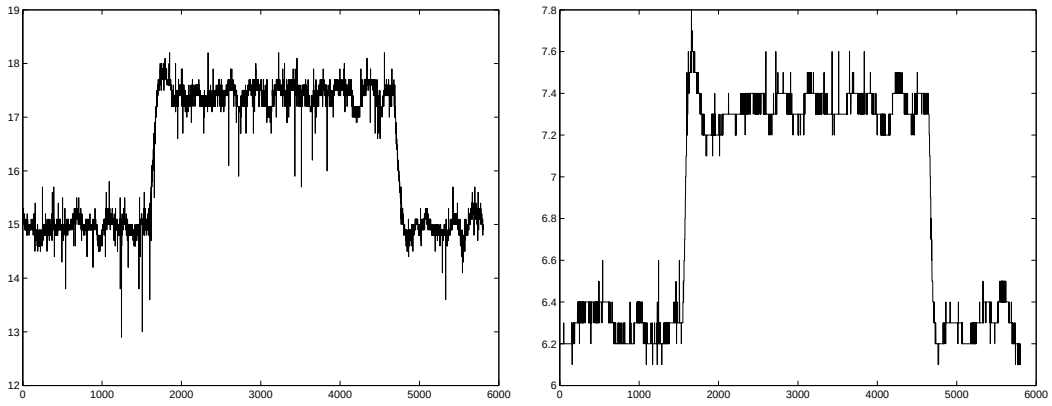


Figure F.2: Step response from the tunnel with a PID controller. (a) shows the step response and (b) the corresponding control signal.

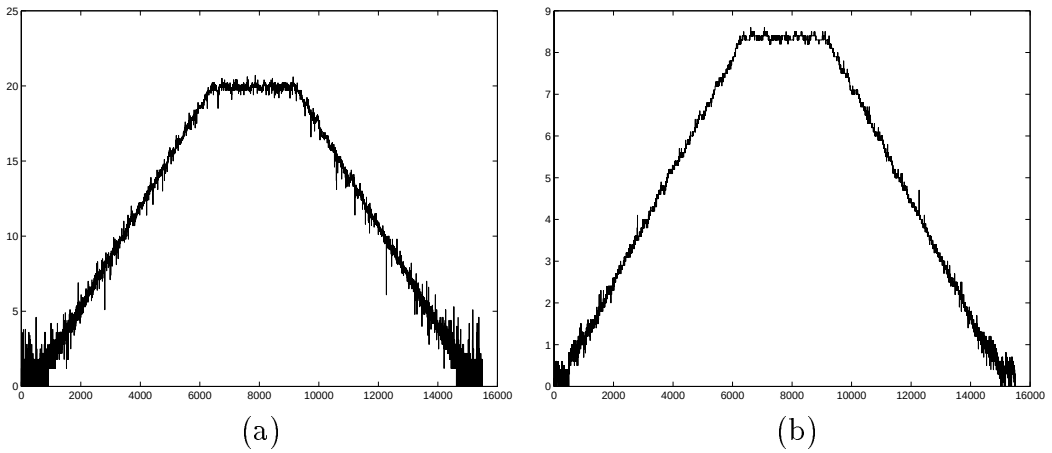
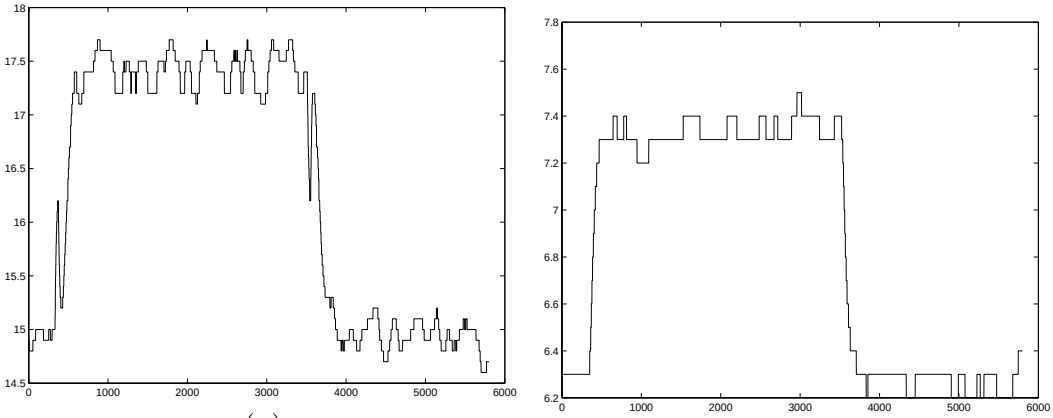


Figure F.3: Ramp response from the tunnel with a PID controller. (a) shows the ramp response and (b) the corresponding control signal.

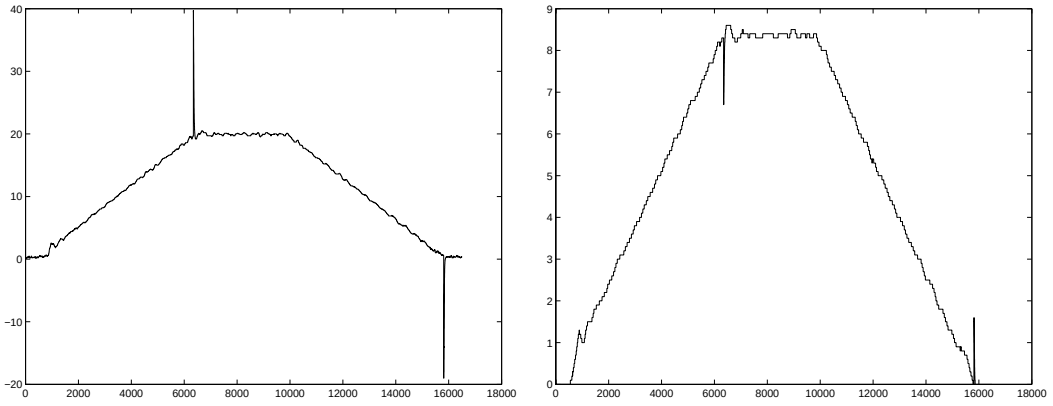
F.2 PID Controller with Filter

The PID controller with filter is slower than the PID without filter and therefore it was possible to make the tests with the controller design found in section 6.4. The results from the tests are shown in figure F.4 (step input) and F.5 (ramp input), respectively.



(a) (b)

Figure F.4: Step response from the tunnel regulated with the PID controller with filter. (a) shows the step response and (b) the corresponding control signal.

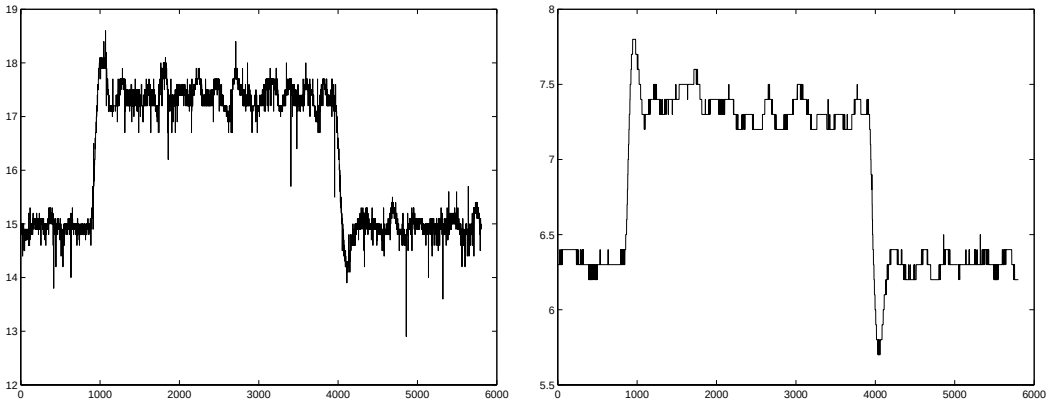


(a) (b)

Figure F.5: Ramp response from the tunnel regulated with the PID controller with filter. (a) shows the ramp response and (b) the corresponding control signal.

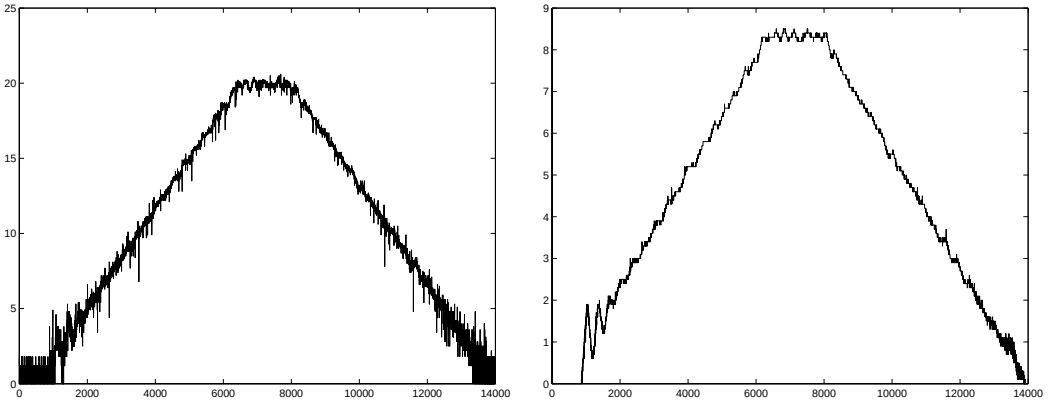
F.3 State-Space

As the PID controller without filter, the simulated state-space controller turned out to be too aggressive, and therefore it had to be detuned. Because of the errors in the wind tunnel model it was not possible to get good results with the state-space controller. The results are shown in figure F.6 and F.7 where the poles for the system are placed in -3, -4 and -4.1. These were found with a hand-tuning as with the PID controller. The estimator poles have been placed in -10 and -11.



(a) (a)

Figure F.6: Responses with the state-space controller regulating the tunnel. (a) shows the response to a step and (b) the corresponding control signal.



(a) (b)

Figure F.7: Responses with the state-space controller regulating the tunnel. (a) shows the response to a ramp input and (b) corresponding control signal.

Appendix G

Component List

The components used are listed in table G.1 The schematic and board layout for use of these components follows in the next appendices.

Component	Value
R1–R6	1 kΩ
R7–R12	1 kΩ
Ro1	20 kΩ
Ro2	10 kΩ
Ro3	20 kΩ
Ro4	10 kΩ
Rt1	40 kΩ
Rt2	40 kΩ
Rt3	100 Ω
Rt4	226 Ω
Rt5	226 Ω
Rt6	300 kΩ
Rt7	300 kΩ
Op-Amp 1–5	Tlc274cn
Connector	37 pin female for print mount
Connector	20 pin female type IDC
Power Supply	$2 \cdot 15$ Vdc

Table G.1: Components used for the connection unit and the PT100 circuitry.

Appendix H

I/O Card Connector

The 37 pin analogue D-sub female connector and the 20 pin digital output male connector of the I/O card is shown in figure H.1

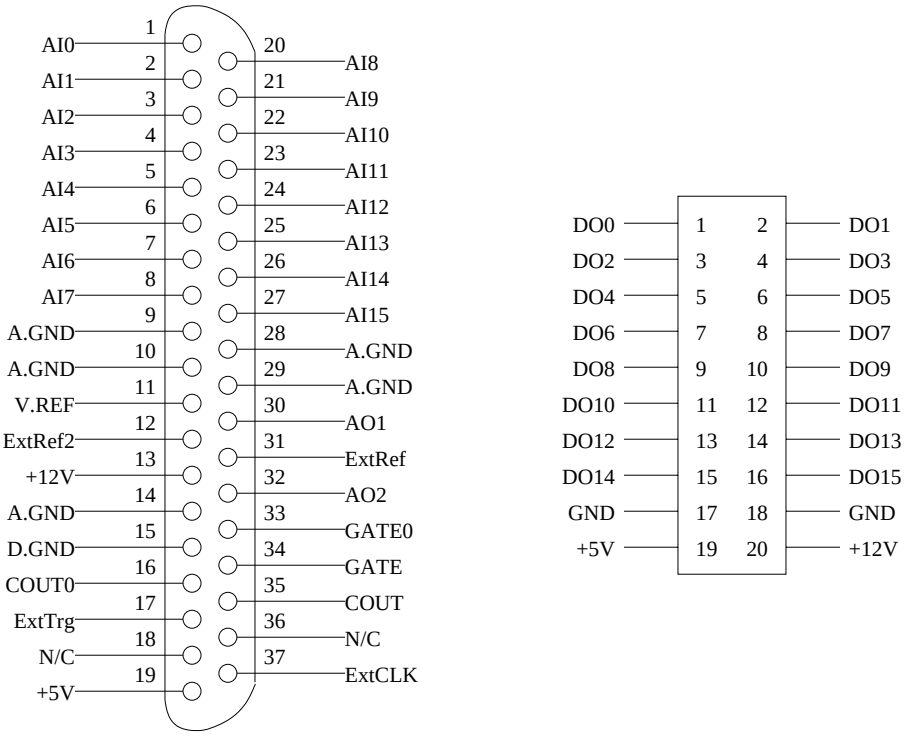


Figure H.1: 37 pin analogue female connector (left) and the 20 pin digital output male connector at the I/O card (right).

Appendix I

Schematic

The schematic for the connection unit is shown in figure I.1 and the schematic for the PT100 circuitry is shown in figure I.2.

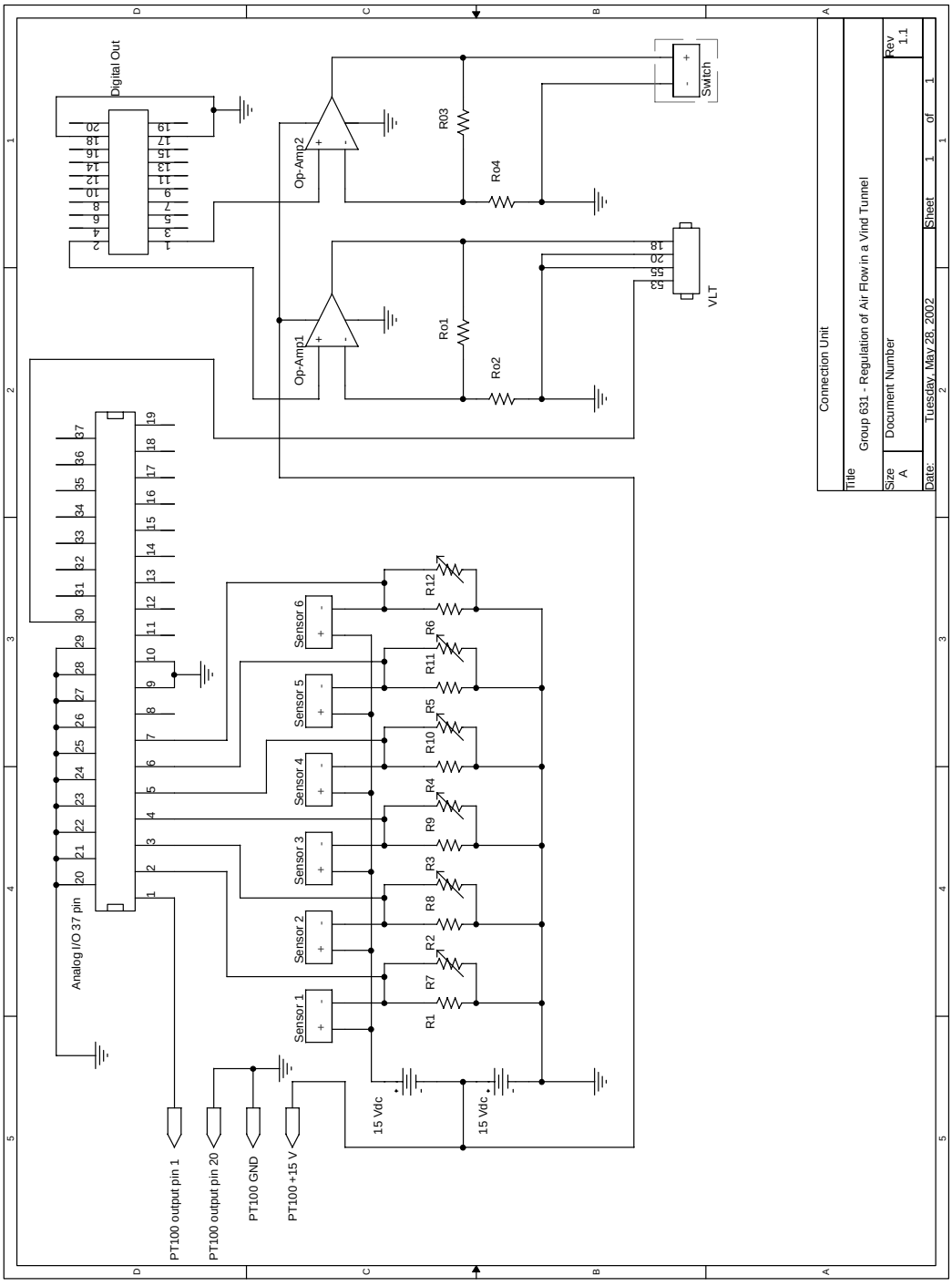


Figure I.1: Schematic of the connection unit.

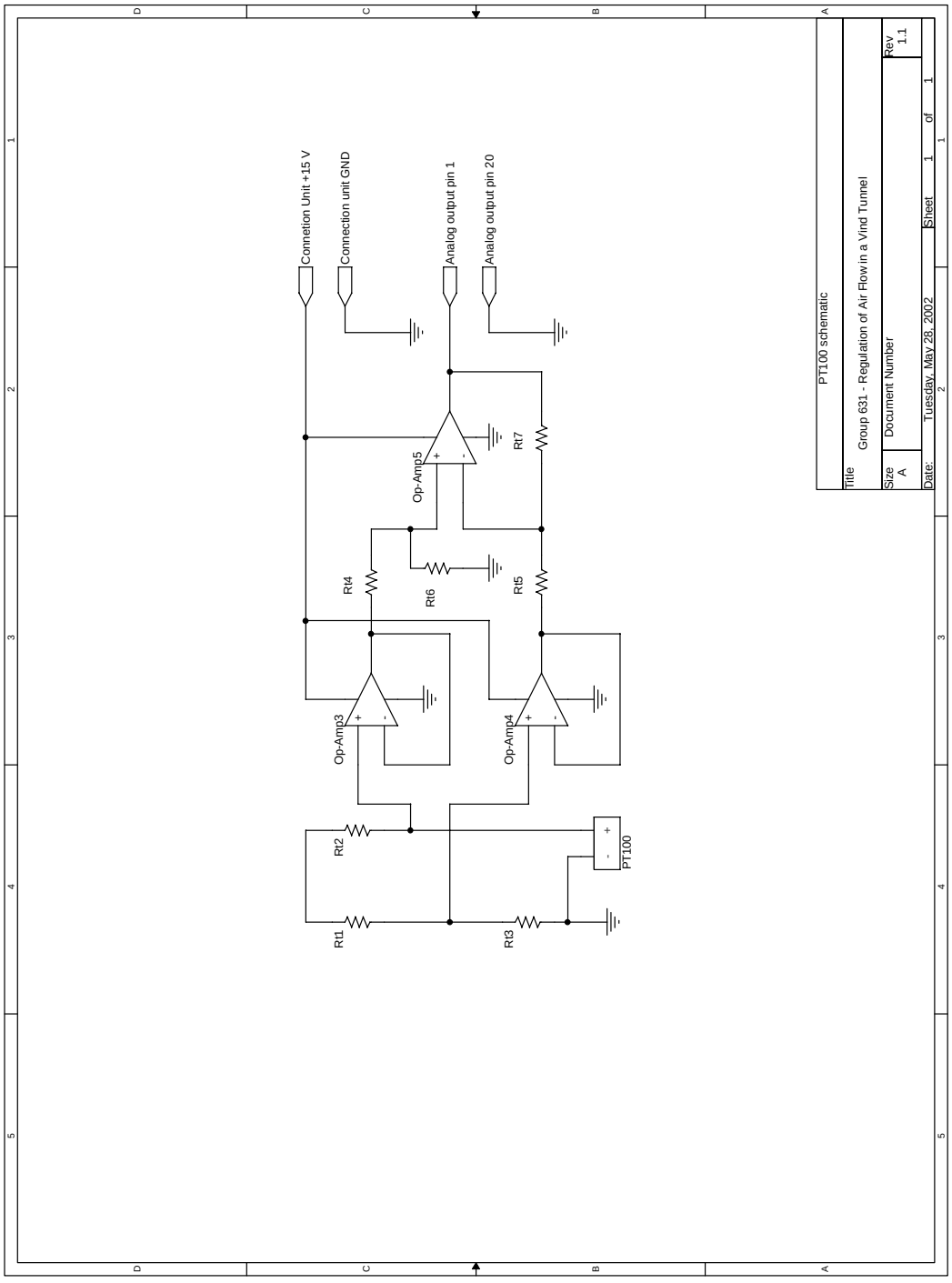


Figure I.2: Schematic of the PT100 connection.

Appendix J

Board Layout

The print layout for the connection unit is shown in figure J.1 and figure J.2. The print layout for the PT100 connection is shown in figure J.3.

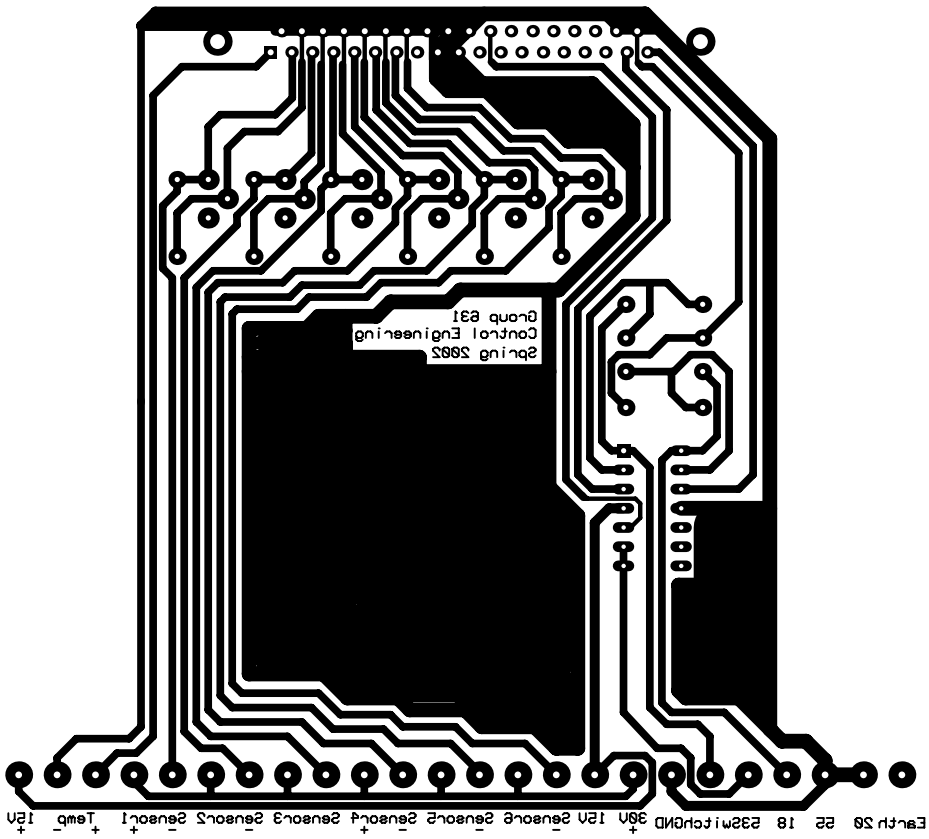


Figure J.1: Print layout for the connection unit. 1:1

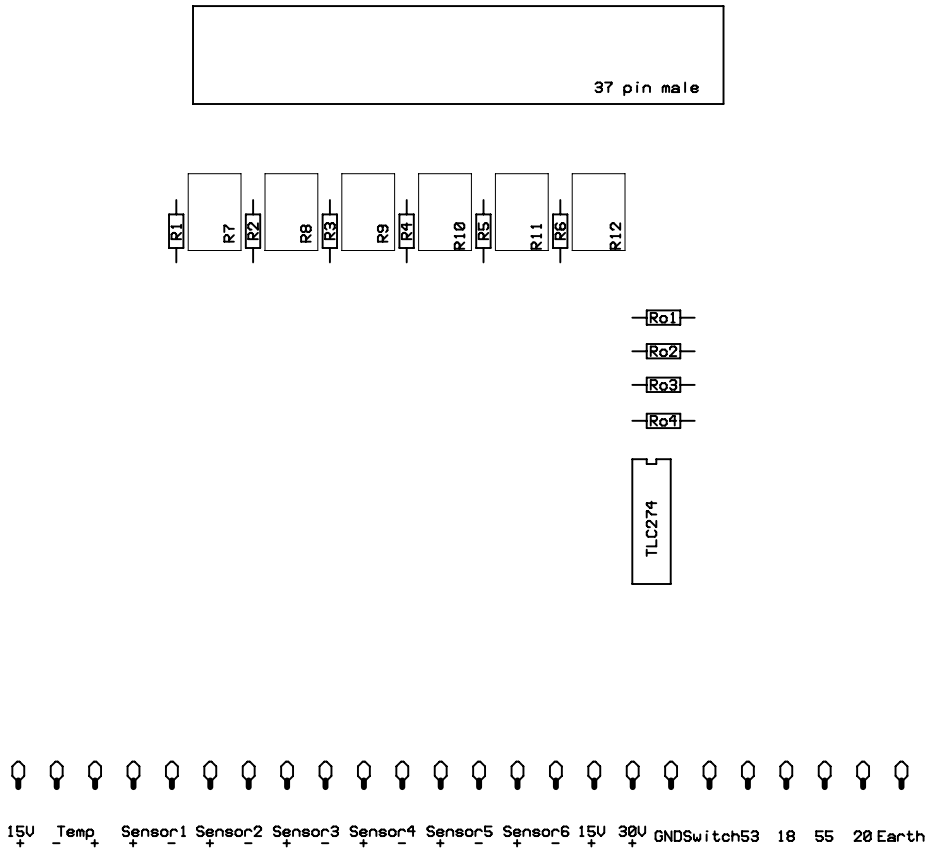


Figure J.2: Component placement.

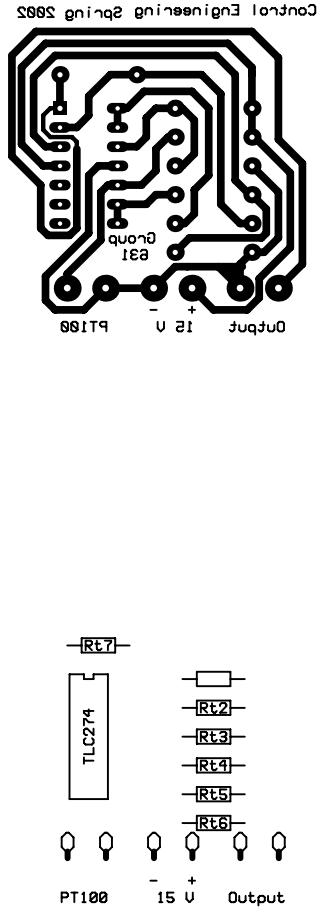


Figure J.3: Print layout for the PT100 connection. 1:1

Appendix K

CDROM contents

- Project Report
- Data Sheets
- Software
  - PelecanWare 1.0
  - PelecanPlot
- Schematics
- User's Manual
- Simulations
- Links
- Abstract
- The Project Group