Realisation of a Realtime Position Determination Mechanism

Kees Lemmens

Faculty of Technical Mathematics
and Informatics,
University of Technology Delft.

November 29, 1994

Contents

Abstract

In this report a description is presented concerning the realisation of a realtime position determination for a special aeroelastic oscillator.
This oscillator is an experimental setup for the study of wind-induced oscillations (galloping) in overhead transmission lines.
The setup consists of a pendulum with a profile attached on one side and counterweights at the other side to keep the setup balanced. The profile is positioned in a windtunnel and the complete system starts oscillating as a result of the external lift- and drag forces induced by the airflow on this profile.
To obtain useful results from the windtunnel experiments it is crucial to have a reliable and accurate device to register the exact position of the oscillator many times per second. This report describes the steps which were made to develop such a Realtime Position Determination Device.

After some disappointing experiments with analogue devices it was decided to develop a digital device, based on the incremental encoders as used in many industrial setups. These incremental encoders have a number of grooves on their circumference and two opto-couplers that produce two 90^o shifted square-wave pulse trains when the system rotates.
These pulse trains are processed with a specially designed electronic interface, based on a special purpose chip made by Texas Instruments: the THCT 2000.

Reading the output produced by this chip requires many control signals that are normally provided through hardware. However, we choose to provide these signals through dedicated software, as this gives the opportunity to change the specifications in a very easy way.
The software (written in C) runs on a simple IBM compatible computer, equipped with an interface card as designed by the Centre for Microcomputer Applications from the University of Amsterdam (the so called UIA/UIB card).

List of Figures

Preface

This report describes the results obtained during a research period Jan-May 1994 embedded in the interfacultary project *Development and design of an aeroelastic oscillator.*

The faculties and groups involved in this project are the Applied Math. Section of the Faculty of Technical Mathematics and Informatics and the Aerodynamics Section of the Faculty of Aerospace Engineering, both at the University of Technology in Delft.
The research period is simultaneously ment as a part of the first degree Physics Course (for teachers) to meet the requirements for the final examination.

I would like to thank the following persons for their support during this research period:

- P.G. Bakker, L.M.M. Boermans, A.H.P. van der Burgh, T.I. Haaker and B.W. van Oudheusden for their valuable advice.
- H.J. Raaymakers for his technical support.
- H. Keus for building a professional final version of the encoder hardware.
- H.X. Lin, H.H. ten Cate and M.R.T. Roest for proofreading the report.
- P.G. van Ditzhuyzen for his final comments on this report.
- R.M.A. van Puffelen for his suggestion to use the THCT2000 chip.

Chapter 1

Introduction

The faculties Technical Mathematics & Informatics (TWI) and Aerospace Engineering (L&R) are working together to study flow-induced oscillations in overhead transmission lines.
Main goal is to study a mechanical model for these oscillations, which makes it possible to observe the influence of several variables and thus helps to understand mechanisms such as instability and limiting behaviour.
Overhead transmission lines tend to build up quite a thick ice layer at temperatures below zero. Combined with a windspeed (from one direction) this ice-accretion will not be circumferential, but will more or less take the shape of an airfoil.
This airfoil may cause instability as a result of the wind-induced lift- and drag forces acting on it.
This instability may result in large amplitude and low frequency oscillations which can cause severe damage to power transmission lines and their supports. For this reason, many devices to avoid these oscillations have been thought of, but none of them has been proven to be really effective thus far.
To study these movements more close an experimental setup was built based on a pendulum with a profile attached to one side.
When this profile is placed in a windfield, the setup will start oscillating as a result of the external forces induced by the windflow.
The mathematical aspects concerned with the modelling the pendulum motions are studied by staff members of the Applied Mathematics Section and the aerodynamical and experimental side is done by the Low Speed Aerodynamics Laboratory of the L&R.

The original type of sensor that was used to determine the displacement of the pendulum (an analogue device) was not very reliable and so a new sensor, based on a digital device, has been developed.

Chapter 2

General descriptions

2.1 Galloping

When a wind-flow crosses a cylinder with a non-symmetric cross section, such as a transmission line or a bridge deck, it is possible that the profile will start moving as a result of the aerodynamic forces acting on the profile. This is determined by the shape of the profile and the amount of damping in the system.
In the case of galloping this motion will be amplified and thus the amplitude will increase until - through increasing limiting forces in the system - a stable situation is reached.

A good description of this phenomenon can be found in [4] :

> Galloping is a low-frequency instability phenomenon of an aerodynamic nature. In practice it may occur on slender, lightly damped structures in cross flow, such as ice-covered transmission lines, high towers or bridge decks. The cause of galloping lies in the occurrence of a negative aerodynamic damping: the aerodynamic forces caused by small relative motions of the structure act to amplify these motions, which results in a growing amplitude. Non-linear effects provide a limiting effect, so that a stable oscillation with a finite but often appreciable amplitude results.

The aerodynamic forces can be split into 2 parts that are perpendicular to each other and to the windfield:

- Lift Forces (L)
- Drag forces (D)

The aerodynamic force component relevant for the motion of the oscillator is the normal force F_y, and this force depends on the incidence angle θ :

$$F_y = L \cos\theta + D \sin\theta \tag{2.1}$$

2.2 Current experimental setup

The setup consists of two horizontal connected bars that can rotate around a centre-of-rotation in the middle of these bars. One side of this system has a profile attached and this profile is positioned in an airflow that is produced by a windtunnel. (see figure 2.1)

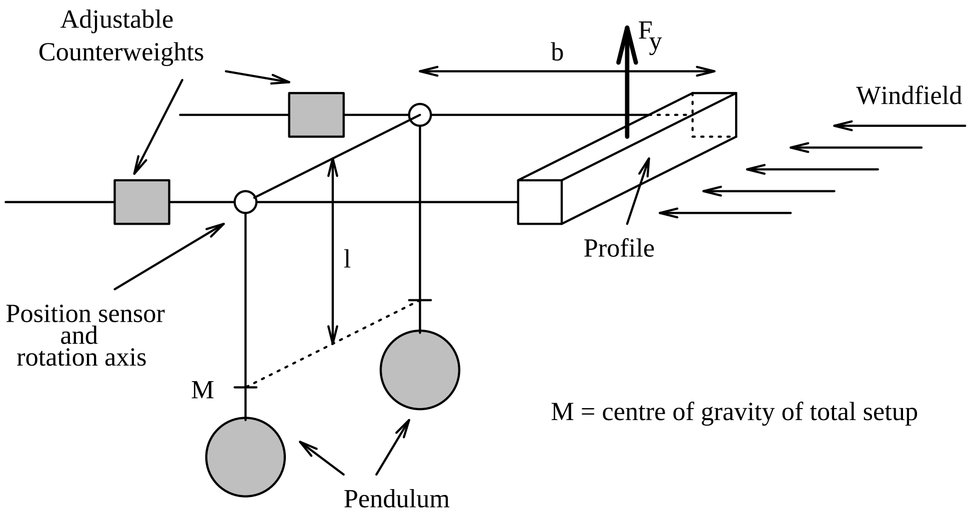

Figure 2.1: Current experimental setup

On the other side of the setup counterweights are mounted to keep the system balanced. To provide a restoring force on the induced movements there is a pendulum attached at the rotation axis. This will turn the system into an oscillator.

Summing all torques for this pendulum yields the following non-linear differential equation:

$$I\frac{d^2\theta}{dt^2} = \lambda\frac{d\theta}{dt} - m \cdot l \cdot g \cdot \sin\theta + F_y b \tag{2.2}$$

With :

I	= the moment of inertia for the complete pendulum
θ	= the angle of incidence
λ	= an arbitrary damping coëfficient
m	= total mass
g	= gravitational constant
l	= length from rotation centre to centre of gravity
b	= length from rotation centre to centre of profile
t	= the time

As the angle of incidence will always be small (no more than approx. 25^o) the whole system can be described by the following differential equation, by assuming $\sin(\theta) \approx \theta$ and rewriting a few terms:

$$\frac{d^2\theta}{dt^2} + 2\zeta_0\omega_0\frac{d\theta}{dt} + \omega_0^2\theta = \frac{F_y b}{I} \tag{2.3}$$

The left hand side of the equation simply describes the free vibrations of the oscillator and the right hand side the vertical acceleration caused by the combination of the lift and drag forces as induced by the airflow.

The external force F_y is the same force as already mentioned in equation 2.1.
In this equation ζ_0 is the damping coëfficient and ω_0 the natural radial frequency.
I, ζ_0 and ω_0 are all assumed to be constant.

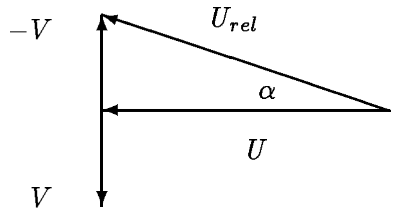

Figure 2.2: Relative windspeed as observed by moving profile

When determining the exact form for the force F_y it is important to realize that - although the windflow U from the tunnel will always be horizontal - the apparent angle of incidence for the windflow will change when the profile has a speed V ($= \frac{d\theta}{dt} \times b$) perpendicular to the windflow, just as is the case with a sailing ship ! (see fig. 2.2)

This implies that in the equilibrium position (where speed is maximum if the system is moving) the angle of incidence for the windflow U_{rel} will not equal 90^o.

For a more accurate description of the aerodynamic forces the reader is referred to [3] and [4].

2.3 Future enhancements

The current setup has only one degree of freedom. This makes mathematical modelling relatively easy, but has the disadvantage that it doesn't resemble the behaviour of galloping power transmission lines very close: galloping transmission lines always involve many degrees of freedom.
However, oscillations with more than one degree of freedom have not been studied experimentally.
Therefor it was decided to extend the oscillator described in this report with a second degree of freedom.

The mathematical modelling for this complex problem is now studied at the faculty of TWI, while the technical realisation is still a matter of discussion.
To support also this new type of oscillator, a Realtime Position Determination Mechanism for 2 sensors was designed and is described later in this report.

Chapter 3

Methods for position determination

3.1 Variable density filters (analogue)

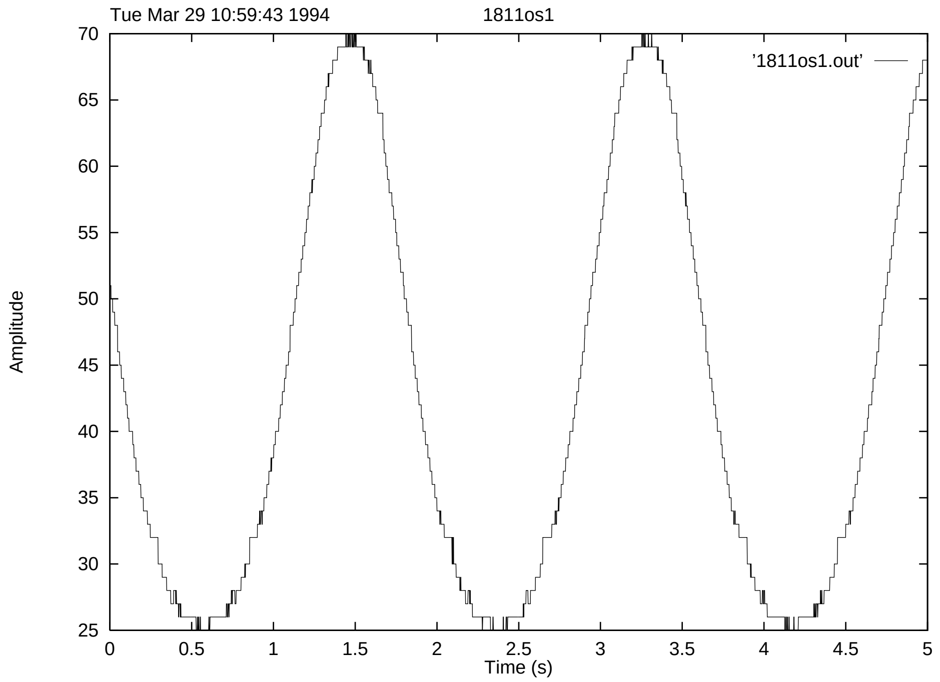

Figure 3.1: Original variable density filter

For the determination of the displacement of the system an analogue device has been used. This device consisted of a small disk with a varying grayscale and an opto-coupler. The permeability of the disk is a linear function of the angle and so the amount of light being received by the fototransistor can be used as a measure for the displacement.
In both pictures 3.1 and 3.2 the non-linearity of the device is clearly visible. This non-linearity proved to be a problem for various reasons, as will be shown later.

A problem with the used opto-coupler (MCA 8) is that it isn't supposed to be used as an analogue device, but only to make clear distinctions between light and dark. This implies that there is a very sharp transition between cut-off and saturation of the foto-transistor.

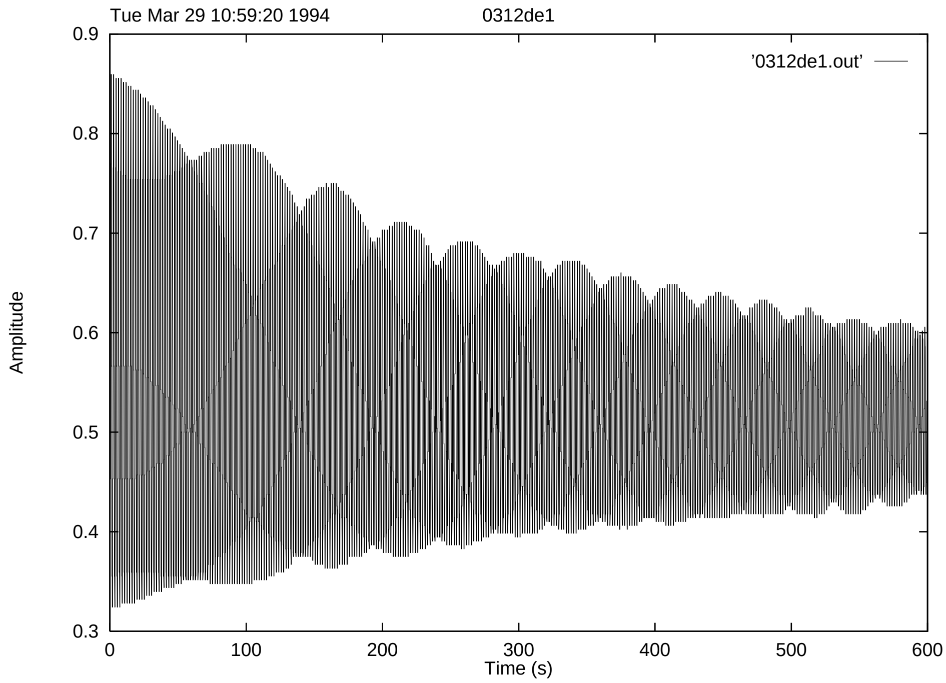

Figure 3.2: Variable density filter: examine system damping

However, in this case the small transition area is used to acquire the data.

Although there are ingeneous techniques to improve the accuracy of this data (as described by B. v. Oudheusden in [4]), this device still doesn't meet the required accuracy.

Another problem with the variable density filter was that it proved to be very sensible for background radiation such as sunlight or TL light. Furthermore it proved to be also quite sensitive for infrared radiation. (λ_{peak} for the MCA 8 is 900 nm)

Figure 3.3 shows what happens when a person comes near the setup: the shadow of the body will change the amount of light being received by the foto-transistor. Also the infrared radiation might play a (small) role.
To show how strong the influence of the background illumination was one measurement with strictly constant conditions (apart from one or two clouds passing the window) was carried out and another one while a person was standing at the light switches, switching the lights on and off during this second measurement.
The results are shown in figure 3.4 and 3.5: it is obvious that the system is much too sensitive for variations in the background light intensity.

3.1.1 Difference signal from 2 opto-couplers

To eliminate the background noise from the analogue encoder output as described above, it was decided to build a second, completely identical opto-coupler with the same response, that would NOT be attached to the moving pendulum.
This one would only register the amount of background radiation. It was supposed that by subtracting this background information from the output signal as supplied by the variable density filter, it should be possible to get a reliable data output.

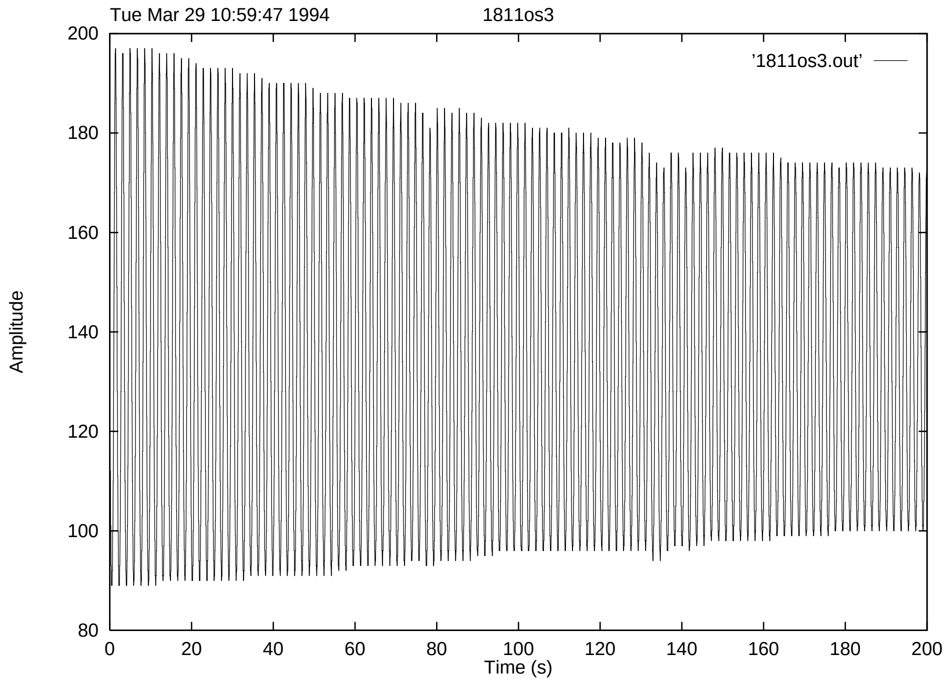

Figure 3.3: Irregularities in variable density filter (noise)

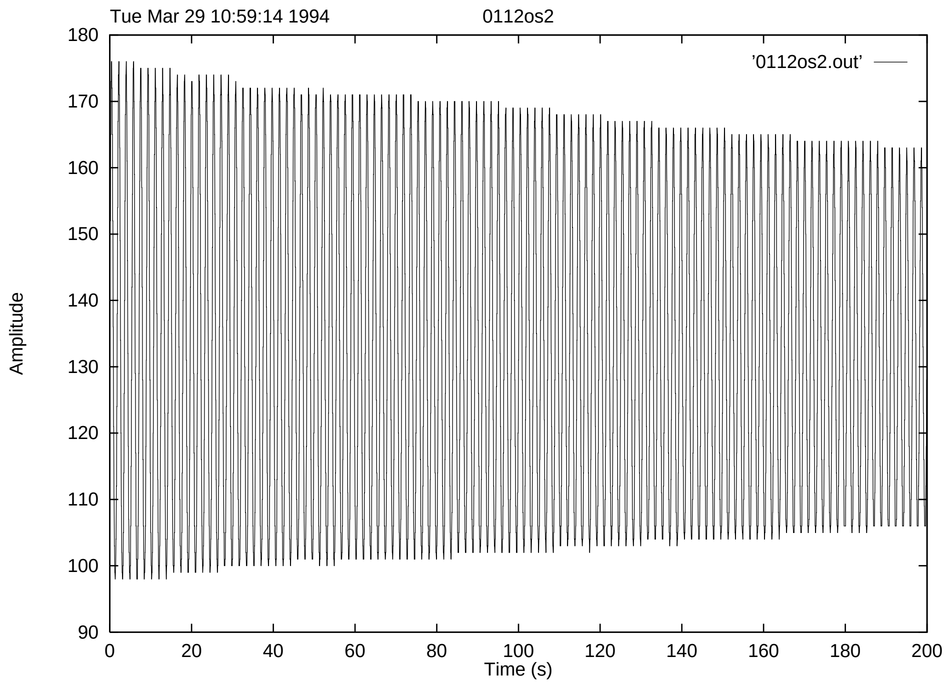

Figure 3.4: Constant illumination

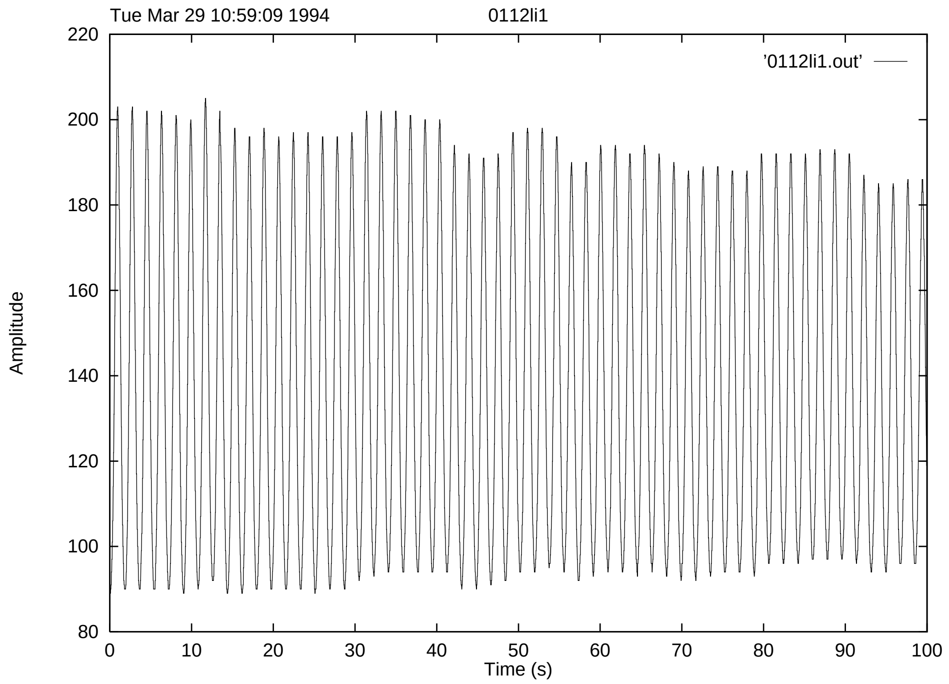

Figure 3.5: Lights periodically switched on and off

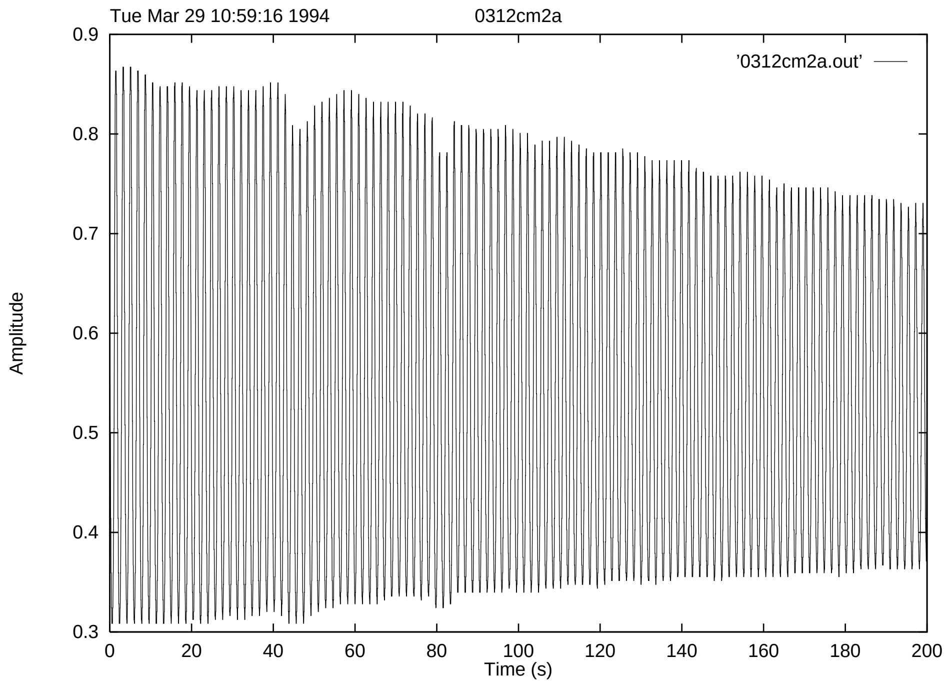

Figure 3.6: Signal from encoder attached to setup

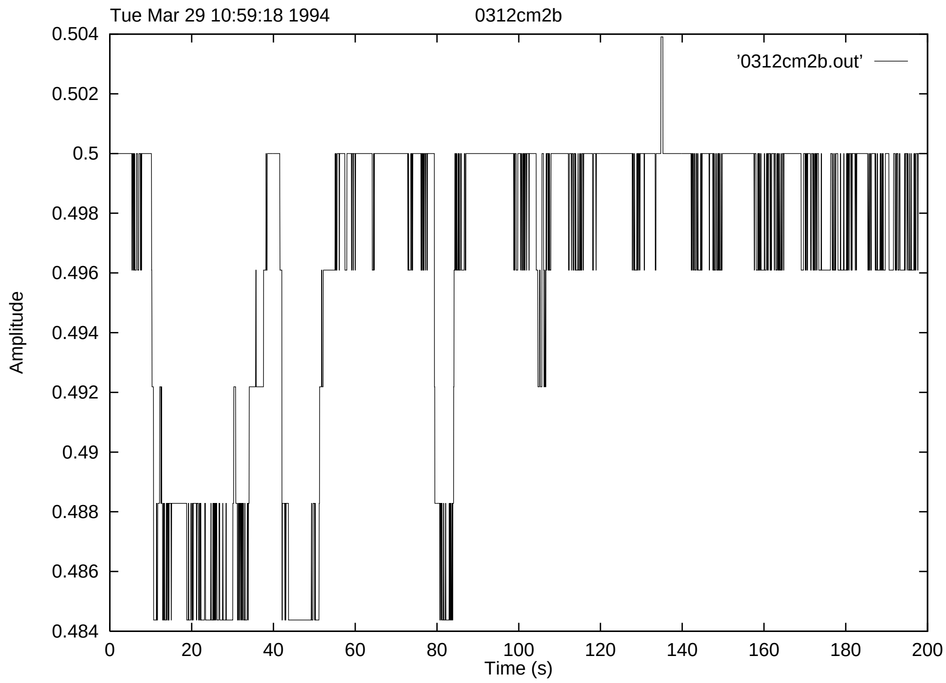

Figure 3.7: Signal from static encoder: only background noise

The two pictures 3.6 and 3.7 show the output signals from the static and dynamic opto-couplers.

The pictures 3.8 and 3.9 show the results when one signal is subtracted from the other in 2 different ways.
If the lower side is more or less corrected the upper side will show larger gaps and if the upper side is corrected by subtracting the noise twice the lower side will be worse than in the original picture.

Although it is possible to manipulate the data in such a way that it is more or less possible to compensate for the irregularities (by making the multiplication factor dependent on the absolute value of the amplitude) it is rather difficult to predict the impact of this manipulation on the accuracy of the system.

Conclusion:
Subtraction doesn't bring any useful results and this is primary caused by the non-linear output signal from the opto-coupler as already described above. It would be possible to think of some more complicated algorithm to subtract one from another to compensate for the non-linearity.
At this point however it was decided to quit this variable density filter and to start the development of another way of Realtime Position Determination.

3.2 Water potentiometer (analogue)

A nice alternative proved to be the "water-potentiometer". In a long basin filled with water and with an electrode at each side, a homogeneous electric field can be created by applying an electric tension between the two electrodes.

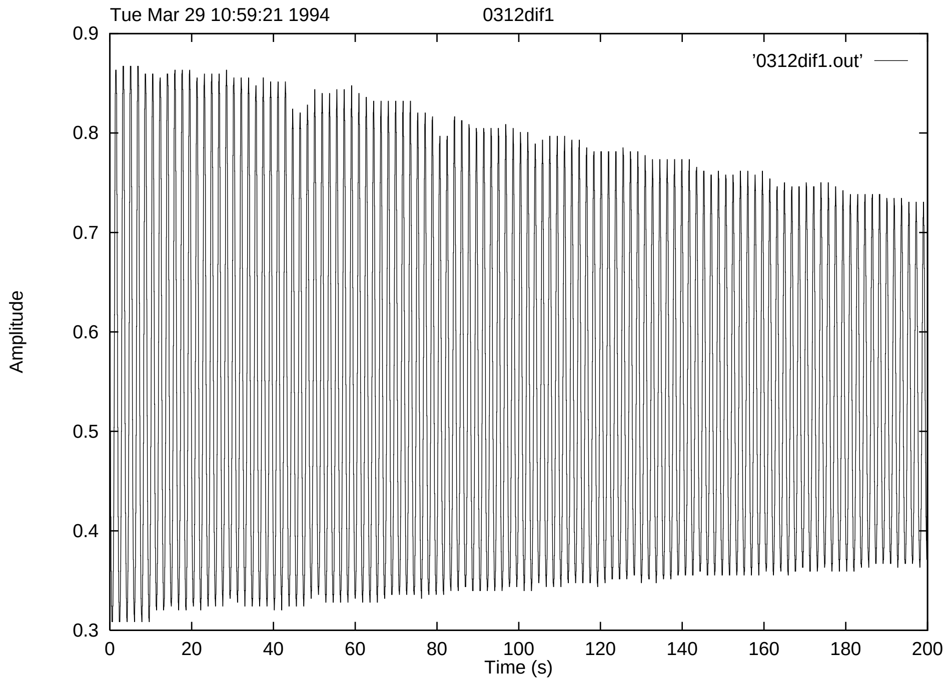

Figure 3.8: Subtract noise from original signal

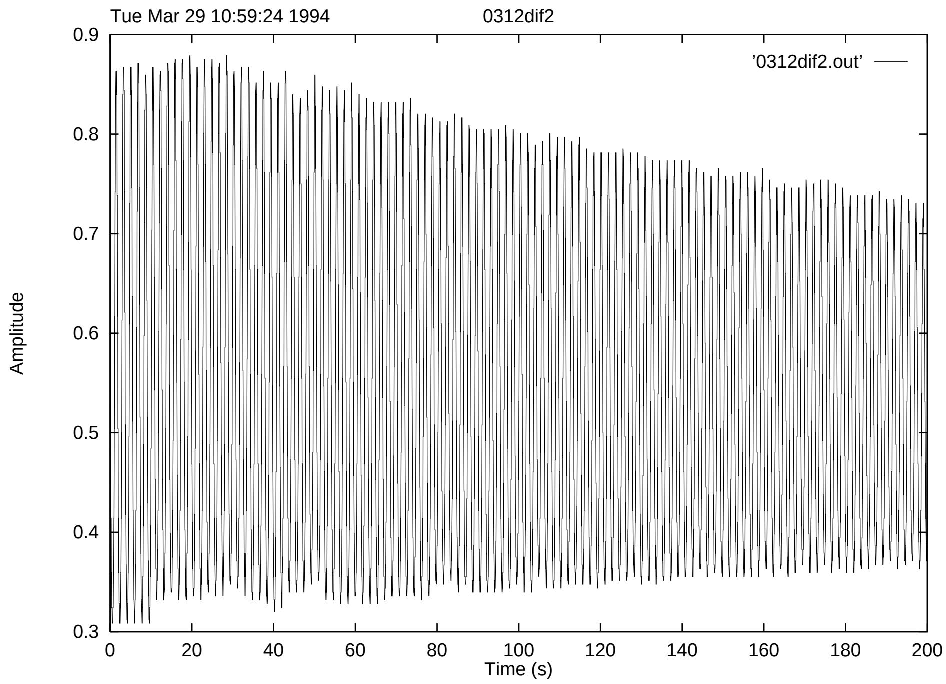

Figure 3.9: Subtract (2 x noise) from original signal

If a pin is attached to the setup in such a way that it can move freely through this basin, then the tension between one of the electrodes and the pin is a measure for the displacement.
Although this doesn't sound very reliable, some quite impressive results were obtained with this technique.

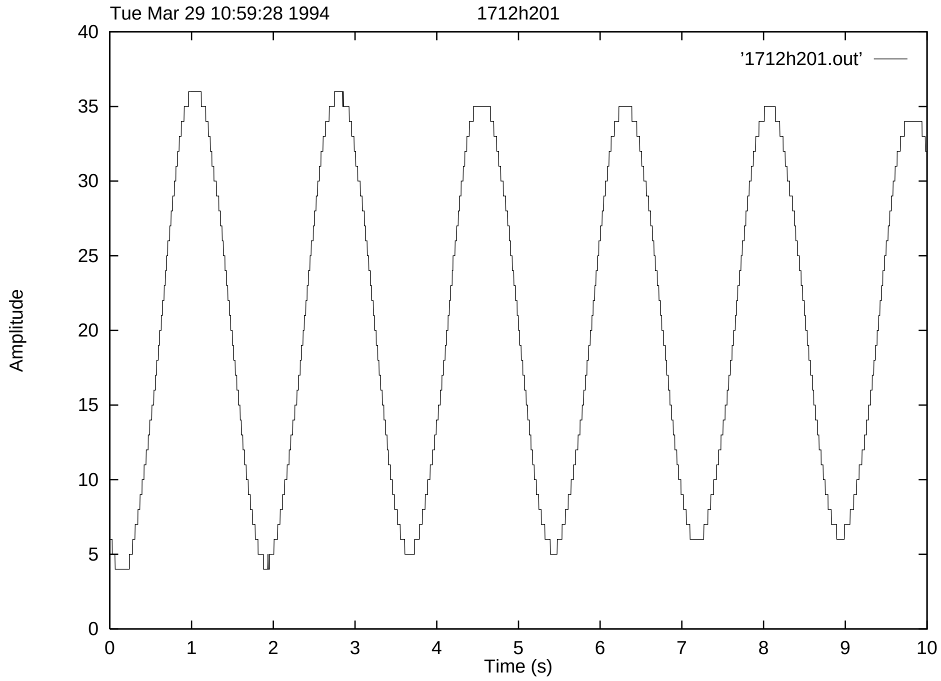

Figure 3.10: Water potentiometer: long bar attached to one side

The first attempt was with a long bar (approx. 30 cm) attached at one side of the rotation centre (fig. 3.10), but this caused to much system damping and - because of the high velocity of the bar - also turbulence in the basin.

Then a very short pin (approx. 2 cm) was attached in the middle of the rotation axis and this seemed to work much better: see figure 3.11.

The mean value of the output signal was a bit higher than 1 Volt, so it was not possible to use the most accurate AD channel (0-1 Volt) on our computersystem.
To fix this the output voltage was lowered with a second, negative power supply, but unfortunately this supply proved to produce a large (50 Hz) hum on the DC output signal.
This hum is clearly visible in figure 3.11: especially at the extremities. However, this little problem could be easily fixed later if it would be decided to continue with this technique. (i.e. get a better powersupply)
If the picture with the short pin (fig. 3.12) is compared with the system damping we gained with the variable density filter (fig. 3.2), which is frictionless, it is clear that the friction caused by the moving pin in the water is almost negligible.

The non-linearity in the pictures is caused by the fact that a short basin and very small electrodes (copper wires) were used. For this reason the electric field isn't completely homogeneous, especially at the edges (close to the electrodes).

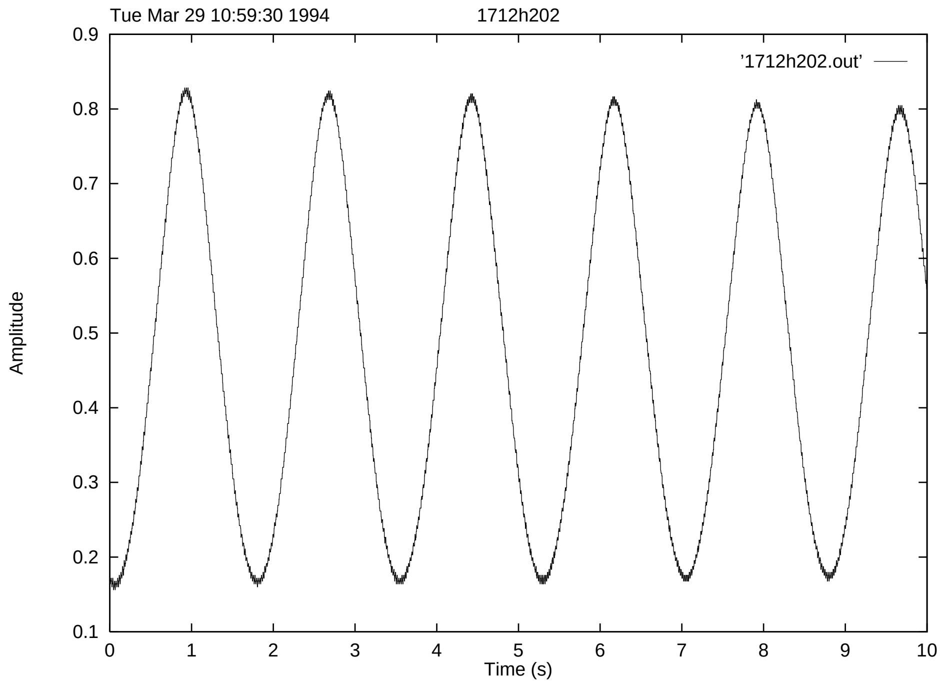

Figure 3.11: Water potentiometer: short pin in the middle of rotation axis

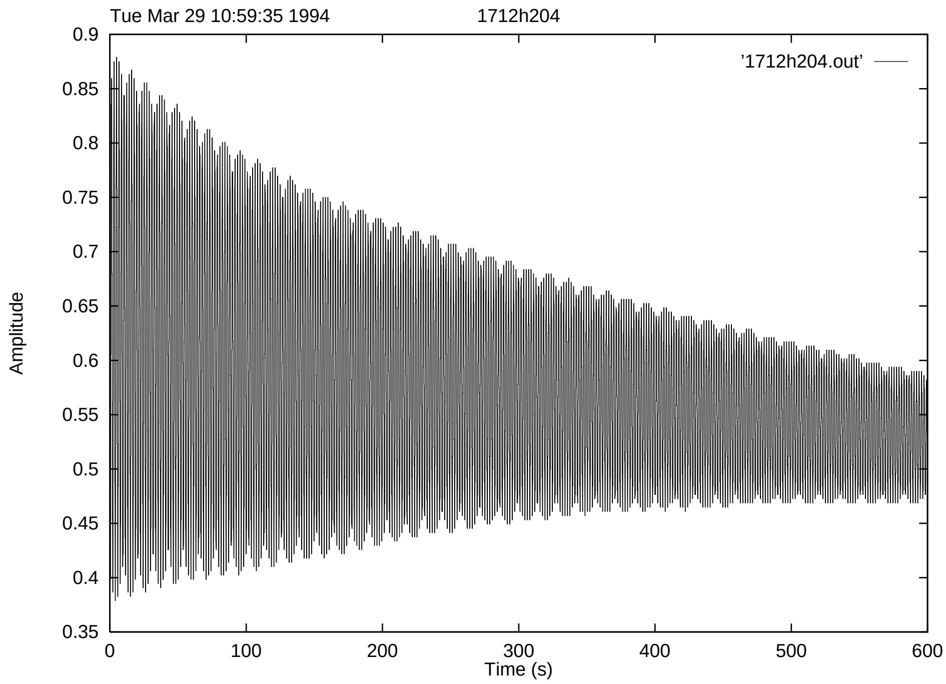

Figure 3.12: Water potentiometer: examine system damping

This could be easily improved (linearised), but a water potentiometer has the disadvantage that it can only be used if the basin is in a horizontal position.
This makes it useless for the extension to the system with two degrees of freedom that is currently being considered and so it was decided to look for yet another solution.

3.3 Incremental encoders (digital)

The results from the analogue encoders as described above showed them to be unsatisfactory. For this reason it was decided to design a complete new digital system, based on an incremental opto-electric encoder disk (attached to one side of the rotation axis) and a pulse counter.

3.3.1 General description

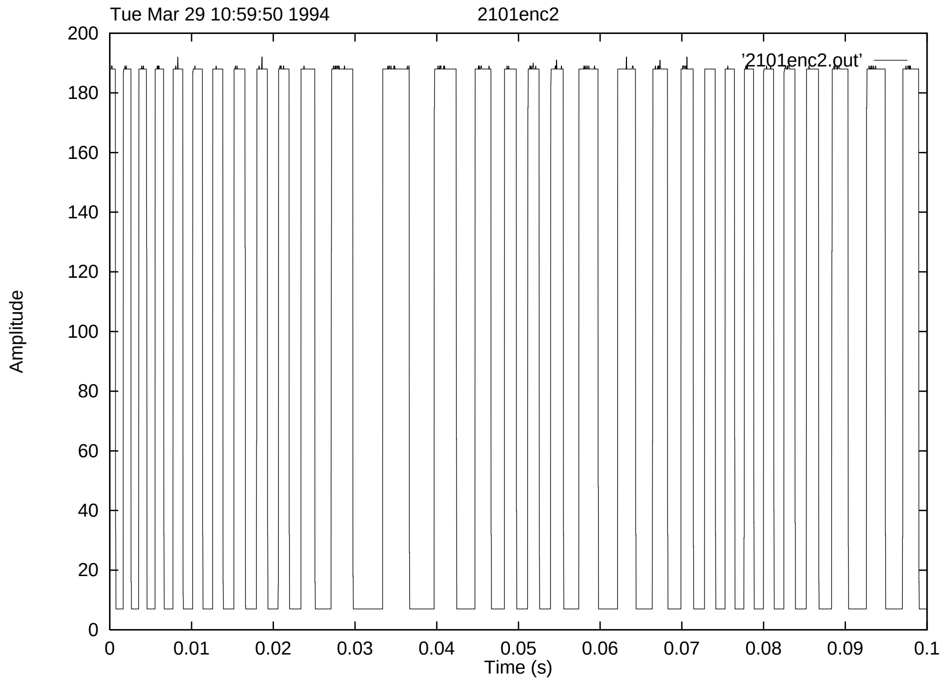

Figure 3.13: Heidehahn ROD426 incremental encoder: output signal of one LED

The following premises were our starting point:

- Maximum amplitude $\pm 30^o \rightarrow 1$ full swing 60^o
- System frequency approx. 0.5 Hz
- Required Accuracy approx. $0.1^o - 0.3^o$

The chosen incremental encoder disk is the ROD426 as produced by Fa. Heidehahn. This encoder has two LED based opto-couplers that produce output pulses shifted by 90^o and it has 1024 grooves over the complete 360^o circumference.
The digital incremental encoder gives the possibility to predict all errors in a very accurate way (it simply depends on the number of grooves) and at the same time eliminates all background noise problems.

Furthermore the encoder output will be always completely linear and it will also be easy to increase the accuracy in the future if necessary.

When the system moves, every time a groove passes the opto-couplers a pulse will appear on the output of one opto-coupler and shortly after that another pulse on the second opto-coupler output. (see fig. 3.13)
By using these two LED output signals it is possible to determine whether we are turning clockwise or anticlockwise.

Example :

Suppose LED 1 goes from 0 to 1 **after** LED 2 goes from 0 to 1 when turning anticlockwise. After some periods we suddenly see LED 2 going from 0 to 1 **before** LED1 goes from 0 to 1.
This can only be possible when the direction of the rotation has altered. (see also fig. 3.14)
This information about the direction is of course absolutely necessary if a counter is used to determine the position: the counter has to know whether it should count up or count down when new data is received.

If a digital counter is used to count the number of pulses received from the two couplers's and if the **order** in which it sees the raising flanks of both pulses is used to determine whether it has to add or subtract from the previous countervalue, this will result in an accurate device to determine the displacement of the moving system.

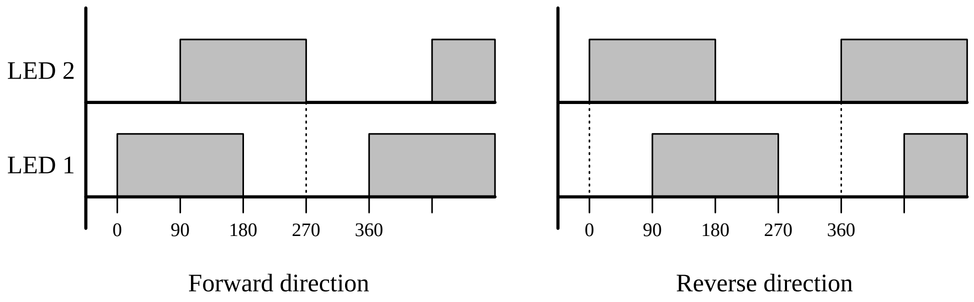

Figure 3.14: Pulse trains for forward and reverse rotation

From the premises above it is easy to conclude that with the current experimental setup only the following number of grooves is used :

$$1024 \cdot \frac{2 \cdot 30}{360} = 170 \text{ grooves} \tag{3.1}$$

With the so-called **quadruple method** the accuracy can be improved if necessary by using the flanks of the output signal instead of the signal itself.
The latter will improve accuracy with a factor 2 for a single output signal, but because there are two LED's producing output signals shifted by 90° in the incremental encoder above there are 4 flanks for one 360° period, so the total accuracy improvement will be a factor 4.

The quadruple method yields for a complete swing from one extreme to the other :

$$170 \cdot 4 = 680 \text{ pulses or : } 2^9 + 2^7 + 2^5 + 2^3 = 1010101000 \tag{3.2}$$

In other words: at least a 10-bit digital input on the sampling device is necessary if it is supposed to use the maximum possible accuracy with this ROD426 encoder.
With a setup that supports all the requirements above, the accuracy can be assumed to be approximately :

$$\text{Accuracy} : \frac{2 \cdot 30\, degrees}{680\, pulses} = 0.088^o \tag{3.3}$$

Much of the functionality described above is integrated into one chip produced by Texas Instruments: the **THCT 2000** (often used for "mouses" or pointing devices). For this reason it was decided to design the interface around this chip.
This THCT 2000 can count the flanks from the 2-bit input and produce a 16-bit output signal. The chip also takes into account the direction of rotation, so it can increment as well as decrement the counter output.

Furthermore a computer interface with at least a 10 bit input is needed to be able to use all information supplied by the encoder. This interface should also be able to send output signals to control the THCT2000 behaviour.

3.3.2 Restrict output signal to 8-bit

If for some reason 8-bit will be the maximum possible resolution, which will be the case if 2 encoders should be read on one interface card, then there are two possibilities :

- Decrease accuracy

  The accuracy can be decreased while retaining the maximum allowed amplitude of 30^o. In this case counting the single pulses will be sufficient, as there will be only 170 pulses/swing as shown above.

- Decrease maximum amplitude

  The maximum allowed amplitude can be restricted to keep the same accuracy.

  Premise : 8 bits must be sufficient, so a maximum of $2^8 = 256$ pulses for one full swing is allowed.

  $$A_{max} = \frac{256 \cdot 360}{res \cdot 2 \cdot 4} \tag{3.4}$$

  The division by 4 is necessary as the quadruple mode is used (4 flanks/per period) and the division by 2 is because a full swing is $2 \times A_{max}$.

  This implies for a 1024 period encoder :

  $$A_{max} = \frac{256 \cdot 360}{1024 \cdot 2 \cdot 4} = \frac{360}{32} = 11.25^o \tag{3.5}$$

  $$\text{Accuracy} : \frac{2 \cdot 11.25}{256} = 0.088^o \tag{3.6}$$

  And this is of course the same as found in equation 3.3.

  However, 11.25^o won't of course be sufficient, so if it is really necessary to work with only 8-bit it is better to drop some of the accuracy.

3.3.3 Reading 2 encoders at the same time

If two encoders should be read simultaneously, as would be the case when experimenting with two degrees of freedom, then there is a problem because the interface cards (both UIA and UIB) only have 16 digital input/output channels as the absolute maximum, which isn't enough for 2 10-bit encoders plus control signals.
There are several ways to solve this problem :

- Use 2 DAC's and 2 analogue input channels on the ADC

  As there are four multiplexed analogue inputs on the measuring-panel this is an easy way to read the signals. If the signals are amplified to use the full 16 bits on the DAC then we wouldn't even loose much accuracy either. However, this requires at any rate a 10-bit ADC (i.e. UIB card) if a 10-bit accuracy is to be achieved.

  It must be admitted that it isn't very elegant to convert a digital signal first into an analogue signal and then back to digital again but if it is a cheap and working solution there is little to say against this technique.

- Use 2 separate interface cards

  This is a much more elegant but also more expensive solution (if there are enough free slots in the computer). However, if money is no problem this certainly is the best option.

  Note that the software on the computer should be capable of reading the two interface cards fast enough, although this won't be a serious problem when about 30 samples per second are considered. (60 samples in total for two encoders.)

- Use a multiplexer on the 16-bit interface card

  As it is only necessary to read about 30 samples per second, there is enough time to use a multiplexer to put both encoders on the same digital input lines on one single interface card. This is however much more complicated as the computer should be synchronised with the multiplexer to be able to determine which encoder is being monitored at a certain moment.

Chapter 4

Hardware descriptions

4.1 THCT 2000 counter

The THCT 2000 incremental encoder interface chip can determine the direction and displacement of a mechanical device based on two input channels from transducers in quadrature. It includes a 16-bit counter which may be used separately.
It can be cascaded to provide accuracy greater than 16-bits and it is designed for use in many types of microprocessor-based systems. Most In/Output lines are TTL compatible.

It may be used in many diverse applications, including robotics, tracker balls (or mouses), tooling machines, conveyor belts and transport mechanisms.

There are 5 major functional blocks in the THCT 2000 :

1. 16-bit Counter

   The counter consists of 2 8-bit counters in cascade and simply counts up or down when the measurement unit generates the appropriate pulses. Carry and Borrow are only used when the complete chip is used in cascade with other THCT 2000's.

2. Measurement logic

   This block processes the 2 input signals from the sensor and generates up and downcount pulses, depending on the flank order of the input signals U1 and U2. (see also figure 3.14 A special direction discriminator that is part of this block controls this functionality.

   It can be controlled through lines M0-M2 : they determine in our application the accuracy of the chip. (single, double or quadruple mode).

M2	M1	M0	description
0	0	1	Single count mode using U1 rising edge (for upcount)
0	1	0	Single count mode using U2 rising edge (for upcount)
0	1	1	Double count mode using U1 rising and falling edge
1	0	0	Double count mode using U2 rising and falling edge
1	0	1	Quadruple count mode using all 4 edges

3. Control logic

Figure 4.1: Block scheme of the THCT 2000

This part receives or transmits the control signals from or to the microprocessor. **RD** is used to move the current value of the counter into the 16-bit latch for later use and **WE** is used to preset a value from the latch into the counter.

A0 is a toggle to determine if we want to read the LSB or MSB of the 16-bit latched value. This is necessary because there is only an 8-bit databus.

READY is a line that should be monitored by the microprocessor: if this line goes low it means that the value on the databus is valid. As long as it is high the value on the databus is undefined.

CS should always be set to 0 before any read or write operation and should be cleared to 1 afterwards.

4. 16-bit latch

   As soon as the Control logic receives a **RD** instruction, the current countervalue is copied into this register. After that this value can be read at any time: from this point there is no need to worry about timing.

5. 8-bit Output register & Multiplexer

   This is the databus where the microprocessor can read the latched countervalues. A problem concerned with this databus is that it is only 8-bit wide. This implies that if the countervalue exceeds 255, two read operations should be used to obtain the complete value in the 16-bit latch.

   If **A0** is 1 the Least Significant Byte is selected through the multiplexer and if it equals 0 the Most Significant Byte.

   For each value a separate **RD** instruction should be given.

Details about the Reset operation and the special characteristics of the **KLI/KLO** and **READY** lines will be given in the section about the incremental encoder interface.

4.2 UIA/UIB PC interface cards

To process encoder signals in a computer some way of interface between the encoder and this computer is necessary.
On the basis of positive earlier experiments, the UIA card made by the "Centre for Microcomputer Applications" in Amsterdam was introduced.

This UIA card (Universal Interface Adapter) has the following characteristics :

- 2 bidirectional 8-bit I/O ports. Port A is always available, Port C is only completely available if no handshaking is required. If handshaking is required then 1 or more bits are lost, depending on the type of handshake used. Port B is for internal use and cannot be accessed externally.
- A 1Mhz clock-oscillator to make the card independent from the PC system clock. (Under MSDOS there is a wide range of clockfrequencies in use nowadays which makes it difficult to use it for a general-purpose card like the UIA.)
- 3 separate 16-bit down-counters (timers), as the internal counter supported by MSDOS PC's is extremely simple (some would even say stupid) and is not programmable in a standard way.
- 4 softswitches to put the timers and the internal clock-oscillator in a cascade.
- A 4 channel 8-bit Analogue-Digital Converter (ADC) with 2 different sensivities : 2 channels up to 1V input and 2 channels up to 5V input. Conversion time required for one sample : 40 μs.
- A 5V power supply capable of max. 100 mA. that can be used for sensors that require an external power supply.

The centre mentioned above also supplies since about a year a new type of card : the so called UIB card.
This card has the same characteristics as the UIA card except for the following:

- The Analogue-Digital Converter now supports a resolution of 12-bits instead of 8-bits.
- The ADC conversion time is now 10 μs for one sample.
- Furthermore the ADC is capable of reading both negative and positive input signals: This implies the sensivity now ranges from -1V to +1V resp. from -5V to +5V.
- The ADC supports a Sample-Hold function.
- The 5V power supply now has a maximum allowed current of 200 mA.

4.3 Incremental encoder interface

To process the quadrature signals obtained from the incremental encoder, hardware based on the THCT 2000 chip from Texas Instruments was used.

Several lines of this chip can be controlled or read through the interface card in the computer.
However, some extra hardware around the chip is necessary to make it work:

- clock

  A very essential part is the clock. According to the documentation the clock speed should be **at least** 4 times greater than the frequency of the quadrature signals.

  With less than 4 clockpulses per quadrature period there is a chance that we'll miss one or more of the 4 pulse edges which will result in a corrupted counter value.

  Average frequency of the setup is 0.5 Hz.

  Number of pulses for a **full** swing :

  $$1024 \cdot \frac{60}{360} \cdot 2 \approx 340 \text{ pulses} \tag{4.1}$$

  This gives us an average pulse frequency of $0.5 \; . \; 340 = 170 \; Hz$.

  The system can be considered as a semi-harmonic system, so we can write:

  $$\int_0^{\pi} f_{max} \cdot \frac{\sin(t)}{\pi} dt = 170 \tag{4.2}$$

  $$f_{max} = \frac{170 \cdot \pi}{\int_0^{\pi} \sin(t) dt} = \frac{170 \cdot \pi}{[-\cos t]_0^{\pi}} = \frac{\pi}{2} \cdot 170 = 267 \; Hz \tag{4.3}$$

  This implies a minimal clock speed of $4 \; . \; 267 = 1070 \; Hz$.

  The clock speed for the NE555 timer can be adjusted according to the following equation:

  $$f = \frac{1}{1.4 \; RC} \tag{4.4}$$

  To be on the safe side a clock speed of 5 kHz is used. This is achieved with i.e. $R = 2.2k\Omega$ and $C = 70nF$. (effectively 4640 Hz)

- M0-M2 switches

  These 3 switches were added to have the possibility to control the accuracy outside the software. (see section about THCT 2000 for more information)

  When thinking of using only 8-bit these switches where important: the accuracy should be always in such mode that we would never overflow the lower 8-bits of our counter.

  The later design however used the software to toggle **A0** and thus reads both LSB and MSB, so from that moment on always the maximum accuracy of 0.088^o could be used.

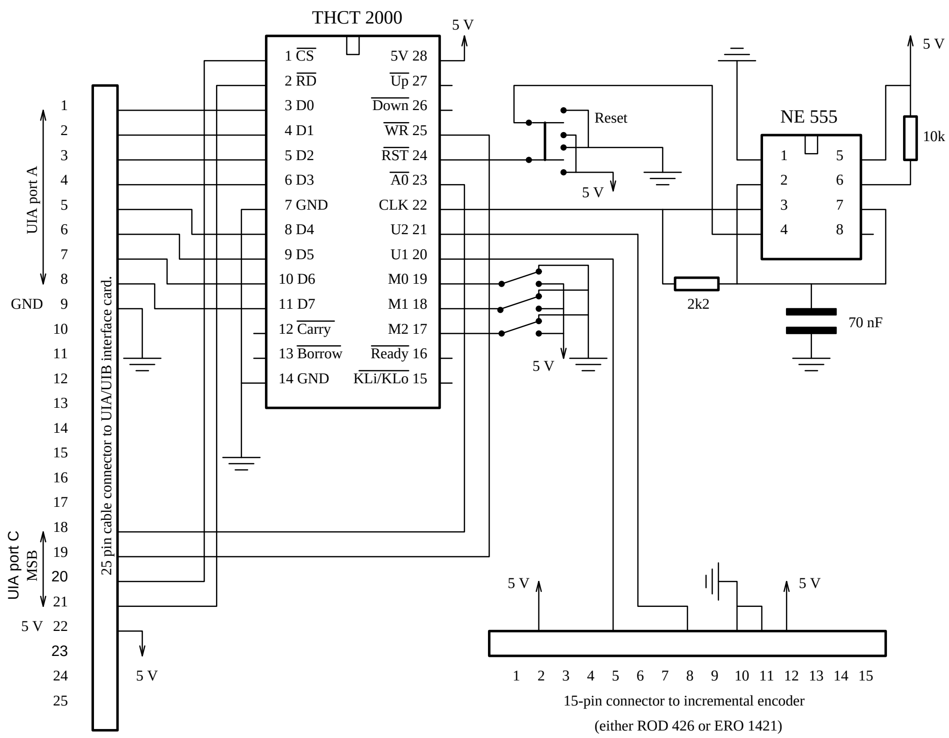

Figure 4.2: Initial design for 16-bit (very unreliable)

- Power supply

  The incremental encoder itself needed about 75 mA. (This was checked with an ammeter) Because the electronics only need about 10 mA, the internal powersupply of the UIA card (100 mA max.) and of course also the UIB card (200 mA max.) would be sufficient.

  This saved a lot of trouble: just plug the interface into the PC and the encoder into the interface and everything should work without external powersupply.

- Unused input lines

  In the first design several unused **input** lines were left unconnected. However, when the first design didn't work it was decided to determine for all inputs exactly whether they should be low or high.

  Of course the unused **output** lines (Carry and Borrow) should never be connected either to 0 or 1 as this could result in shortcircuit !!

  There is something special about **READY** and **KLI/KLO**: they are so-called "open-collector" in/outputs. This implies that they should be **always** connected to 0 or 1 or else the setup will show a very unpredictable behaviour.

  It took about 3 full days to find out why the design in figure 4.2 sometimes worked and sometimes didn't. Some loose wire was suspected and the whole system was reassembled twice before any other explanation was considered ...

  The pull-up resistor on **KLI/KLO** proved to fix the problem.

  Without this pull-up resistor the behaviour of the **KLI/KLO** input would be somewhat like a radio antenna: pointing with one finger to the setup was enough to switch

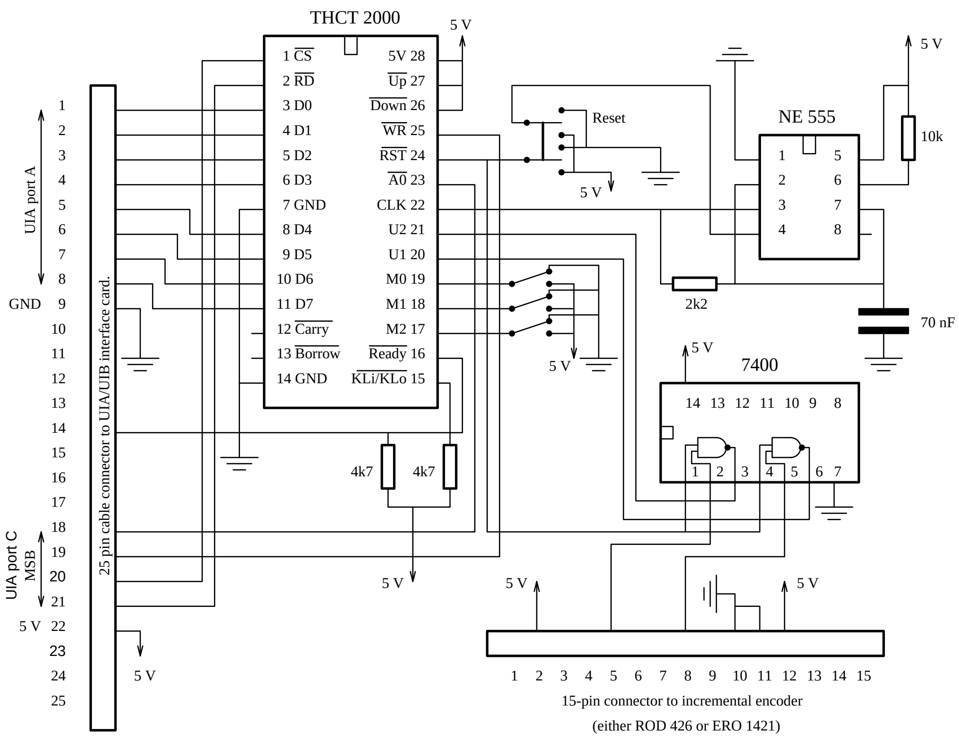

Figure 4.3: Final and working design

from 1 to 0.

Unfortunately the chip blocks completely until the next **RESET** after **KLI/KLO** goes from 1 to 0 for only a short moment. This explains why it took so long to find out: a pull-up resistor was tried several times, but because it didn't seem to make any difference (the chip was already blocked !) we started searching somewhere else.

- Ready line

  Although the chip is very fast, it always takes some time before an instruction would be carried out. For this reason a special **READY** line is available to determine whether or not the chip is ready with the last instruction.

  In the beginning the computer software was considered to be slow enough to forget about monitoring this **READY** line, but because the signal did show some needles now and then, it was decided later to add monitoring this line to the software.

  This immediately fixed the needle problem.

  Note that this line is also an open-collector output, so it also needs a separate pull-up resistor.

- Reset

  A Reset pulse will clear the counter to zero, clears the 4 D-flops used for the direction discriminator and reenables the system after it has blocked for some reason. (i.e. **KLI/KLO** went low)

  To avoid unpredictable behaviour after the reset, it is important to keep **U1** and **U2** high during a reset. For this reason the 2 NAND ports in the U1/U2 lines from the encoder were added.

As long as the reset button is not touched these NAND's will only invert the signals. But as soon as the button is pressed it will make U1 and U2 high, no matter what signals would be generated by the encoder.

Chapter 5

Software descriptions

5.1 IPCOACH description

The IPCOACH program as developed by the CMA group (associated with the University of Amsterdam) is an intersting program to acquire, plot and/or convert data from sensors connected to the analogue input on the UIA/UIB cards.
The software can use several types of calibrations (such as curve-fitting) to compensate for non-linearity in sensors, it can enlarge small parts of plots and it supplies many functions to postprocess acquired data. (i.e. differentiate a signal, remove needles etc.)

The IPCOACH program was used to obtain all output generated by the analogue devices described in this report.
We also used this program as a kind of oscilloscope to check several signals on the hardware during development of the digital sensor.

It was however useless for the digital sensor, because this sensor didn't use the analogue, but the digital in/outputs on the UIA/UIB interface cards.
Furthermore it wasn't possible with the basic software to send control signals to the system and it also wasn't very friendly in writing the acquired data into an ASCII file.
For these reasons special software for the digital sensor was developed.

5.2 Control UIA/UIB card registers

Controlling the UIA/UIB card registers through software is comparatively easy:
Every card has a range of 9-10 8-bit memory addresses mapped on standard memory.
The baseaddress of the card determines where this range will start.
The rest can be found by offsetting the base-address with the appropriate offset.
The base address can be adjusted through a set of jumpers on the UIA/UIB interface cards (either \$100, \$110, \$300 or \$310).

Hereafter follows a listing of the headerfile where all the uia registers are defined.

```
/* uia.h : Kees Lemmens; Feb. 1994 */

#define BASE        0x300  /* interface card base address */

#define PPI_A       (BASE + 0)
#define PPI_C       (BASE + 2)
```

```
#define PPI_CONTROL (BASE + 3)

#define PIT_RWCOUNT1 (BASE + 4)
#define PIT_RWCOUNT2 (BASE + 5)
#define PIT_RWCOUNT3 (BASE + 6)
#define PIT_CONTROL  (BASE + 7)

#define AD_C         (BASE + 8)

/* next are the registers in the THCT 2000 chip that are
   controlled through the 8 bits of PORT C on the UIA card.
*/

#define A0 0x10
#define WR 0x20
#define CS 0x40
#define RD 0x80

#define READY 0x01
```

The only registers used by the software are the **PPI_A**, **PPI_C** and **PPI_CONTROL** registers. To read or write these memory locations under MSDOS and BORLAND-C, two special functions are provided: `inportb()` and `outportb()`.

To manipulate single bits in the register first the original value should be read, then the bit(s) that need to be changed be cleared or set using <OR> or inverted <AND> operations and finally the new value should be written to the register:

```
void set_portc_up(int n)
{
   outportb(PPI_C,inportb(PPI_C) | n);
}

void clr_portc_up(int n)
{
   outportb(PPI_C,inportb(PPI_C) & ~n);
}
```

The **PPI_CONTROL** register is used to determine whether a port is going to be used as input and or as output.
Port C can be split in an upper and lower part that can be configured as in- or output separately. This is important as both input (READY) and output (A0, WR, CS RD) lines to the THCT 2000 chip are used by the software.
Port A can only be configured to be either completely input or completely output. This port is used to read the 8-bit databus from the THCT 2000.

5.3 readuid program

The **readuid** program controls the THCT 2000 behaviour, supplies the appropriate control signals for the hardware, reads the processed encoder data at regular intervals and

stores the sampled values together with a timestamp in a large memory buffer until the mainloop is finished.
After the last sample is acquired, the contents of the memory buffer will be written to an ASCII file and the program will exit.

```
      ***  READUID: program to read port A from UIA card  ***
      Samples the (16-bit) input from the incremental encoder
           by manipulation of the port C output lines.
               v 1.1: (c) Feb 1994; Kees Lemmens

Usage: readuid [-n=<nr>] [-t=<nr>] <file>

  -n   Number of samples (default %u)\n",SAMPLES
  -t   Approx. time between 2 samples (default 100 ms)
  file Send output to <file> (con: sends to screen)
```

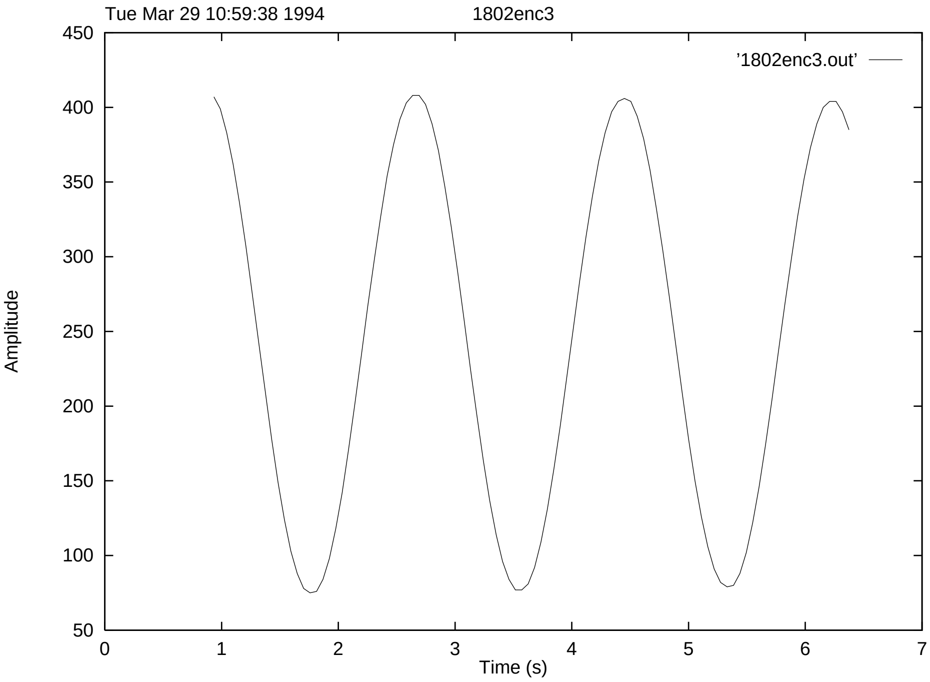

Figure 5.1: Incremental encoder: highest resolution output (1)

A few remarks about the **readuid** program:

- Timer

  When pulling data from the interface it is of course important to have an accurate timer, so to be able to determine exactly at what time a certain sample was acquired.

  This is no problem at all on most computers: most of them have adjustable timers that have timer intervals ranging from only a few μs to many hours.

  But unfortunately the MSDOS BIOS is extremely simple with respect to timers: it only supports 1 (one) fixed timer interval of $3600/65536 \approx 55\ ms$.

According to B. v. Oudheusden ([4]) reading 22 samples per period was sufficient. With a setup frequency of approx. 0.5 Hz this implies that a sample frequency of at least 90 ms is needed.

As this could be done with the simple standard DOS timer, it was decided not to put any effort in using an external timer, such as the timer in the UIA card.

But because it was not possible to sample exactly at every 50 or 100 ms with the DOS timer, it was impossible just to write out the sampled values and assume a fixed timer interval. For this reason a special record to store both the exact sample time and the sample value was introduced:

```
typedef struct _sample { clock_t time; int value; } Sample;
```

This will be the standard format for all datafiles produced by the programs written for this setup. (Only the binary values will be converted to ASCII)

One more thing: as the `clock_t` datatype can only store 32-bit numbers, the maximum sample period is:

$$\frac{2^{32}}{65536/3600} = 2,36.10^{8} \text{ seconds} \tag{5.1}$$

Nevertheless this is not considered to be a serious problem ...

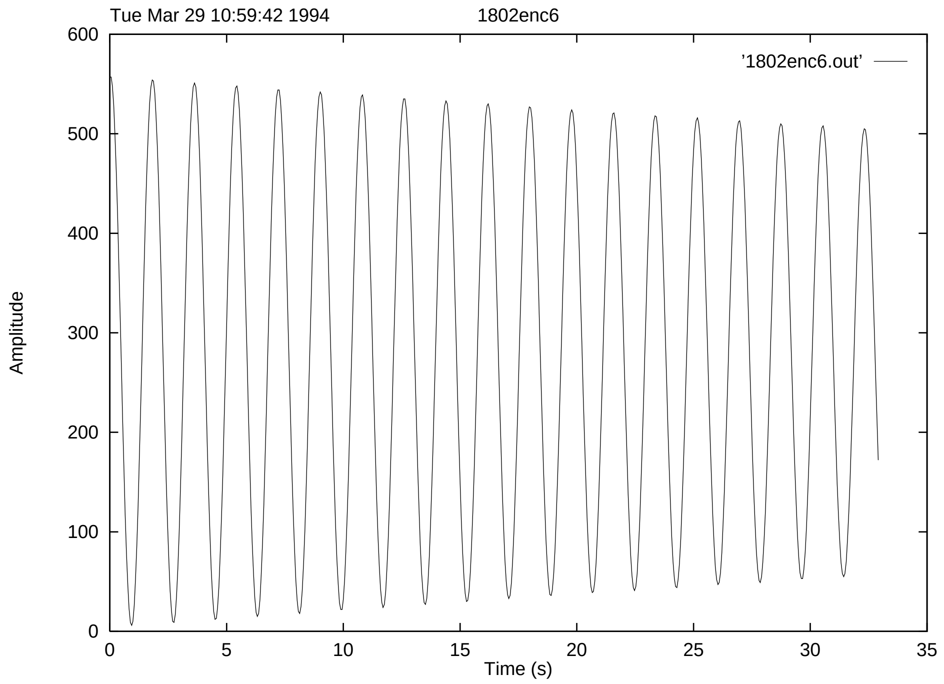

Figure 5.2: Incremental encoder: highest resolution output (2)

- Number of samples

  Because of some limitations in the memory organisation of PC's, it was difficult to use 32-bits indices for arrays. For this reason the maximum number of samples was limited to the maximum possible value for an unsigned integer: $2^{16} = 65536$.

This implies that it is not possible to measure longer than approx. 3600 sec. if using an interval time of 55 ms.

- PPI Control register

  To use port A for input, port C lower for input and port C upper for output a value of 10010001 should be stored in the PPI Control register. (see also the appendix of [7]) This is done in the `select_port_ac()` function.

- Reading port A

  Every time a new value should be read from the encoder, several tasks should be carried out:

  - Set the timestamp.
  - Pull the THCT 2000 CS and RD lines low to store a value in the latch.
  - Select the Low Byte of the latched value through A0.
  - Wait for the Ready line to become high.
  - Read the contents of PPI port A register.
  - Select the High Byte of the latched value through A0.
  - Wait for the Ready line to become high.
  - Multiply PPI port A register with 256 and add to previous value.
  - Reenable the THCT 2000 chip by setting CS, RD, WR and A0 (!!) high.

  This is all done in the `read_porta()` function.

- Memory buffer

  Because it was found that writing a sample to disk immediately after it was read could **not** be done within 55 ms, all samples had to be kept in memory until the sampling process was finished.

  The buffer is created by dynamic allocation of exactly the necessary amount of memory. It is of course also possible to allocate always the maximum amount of memory, but this makes the program much less flexible. Now it can run on every PC - even with little memory - as long as no more samples than will fit in the memory are acquired during one run.

- Progress information

  The program will print the sampled value on the screen as soon as it is read. This is to have some sort of progrees information, provided there is enough time to do so. In the current program this is only done for sample intervals greater than 500 ms. (`#define MIN_REPORT`)

- Wait for next sample time

  The function `waitinterval()` will pause after every sample, until the time for the next sample has arrived. As soon as the interval seems to be too short (current time > next sample time) the program will quit with an error message.

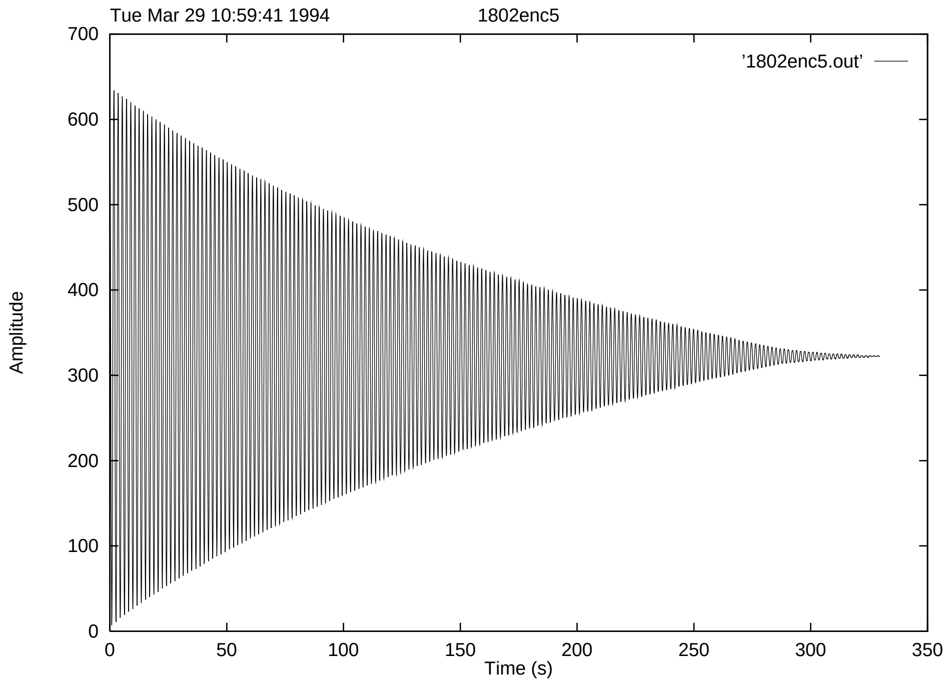

Figure 5.3: Incremental encoder: examine system damping

Figure 5.1, 5.2 and 5.3 show the results of the new Realtime Position Determination setup.

As can be seen from the picture 5.3 the system damping proved to be much higher than the damping found with the variable density filter or the waterpotentiometer. This is caused by the fact that the encoder type used (ROD426) is equipped with a ball bearing.

However, if the damping proves to be too high, this encoder can be easily replaced with a frictionless (but also more vulnerable !) type, i.e. the same as the that is going to be used for the second degree of freedom: Heidehahn ERO 1421.

Hereafter follows a complete listing of the **readuid** program.

```
/* readuid2.c : Kees Lemmens; Feb. 1994.

   Program to read the UIA digital 8-bit inputport (port A) from a PC.
   This program also outputs several control signals to the 8-bit
   in/output port C on this UIA card. These output signals are used to
   manipulate the registers from the THCT 2000 encoder chip.

   With the encoder interface hardware and this program we are able to
   read data with 16-bit resolution, although we can only read 8-bit
   signals from the THCT2000 data register.

*/

#define DEBUGLEVEL 0

#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <time.h>
#include <math.h>
#include <limits.h>

#if (DEBUGLEVEL == 0)
#include <dos.h>
#endif

#include "uia.h"

#define SAMPLES 100
#define DEFTIME 100

#define MIN_REPORT 500

typedef struct _sample { clock_t time; int value; } Sample;

void select_port_ac(void)
{
#if (DEBUGLEVEL == 0)
    outportb(PPI_CONTROL,0x91);   /* 1001 0001 naar control */
#endif
}

#if (DEBUGLEVEL == 0)
void set_portc_up(int n)
{
   outportb(PPI_C,inportb(PPI_C) | n);
}

void clr_portc_up(int n)
{
   outportb(PPI_C,inportb(PPI_C) & ~n);
}
#endif

void wait_for_ready(void)
{
   /* wait for data to be available (READY signal goes low) */

   while(inportb(PPI_C) & READY);
}

void read_porta(Sample *s)
{
       s->time = clock();
```

```
#if (DEBUGLEVEL > 0)
      s->value = (int)2048*sin((double)s->time/200);
#else
      clr_portc_up(CS|RD);     /* pull CS & RD low, WR high (THCT2000) */
      set_portc_up(A0);
      wait_for_ready();
      s->value = (signed)inportb(PPI_A);       /* Low byte */
      clr_portc_up(A0);
      wait_for_ready();
      s->value += (signed)inportb(PPI_A) * 256;  /* High byte */
      set_portc_up(CS|RD|WR|A0);  /* CS & RD & A0 & WR high */
#endif
}

void waitinterval(clock_t newtime)
{   clock_t curtime = clock();

#if (DEBUGLEVEL > 1)
    printf("newtime %ld currenttime %ld\n",newtime,curtime);
#endif

    if(curtime >= newtime)
    {    printf("Interval too short or system too slow !\n");
         exit(1);
    }
    while(clock() < newtime);
}

void usage(void)
{  printf("Usage: readuid [-n=<nr>] [-t=<nr>] <file>\n\n");
   printf("  -n   Number of samples (default %u)\n",SAMPLES);
   printf("  -t   Approx. time between 2 samples (default %ld ms)\n",DEFTIME);
   printf("  file Send output to <file> (con: sends to screen)\n");

   exit(1);
}

void main(int argc,char *argv[])
{
   long count=SAMPLES;
   unsigned x;
   Sample *buffer;
   clock_t starttime; /* to start at zero */
   clock_t interval=DEFTIME;
   FILE *out;
   double factor = CLK_TCK / 1000.0;

   printf("\t*** READUID: program to read port A from UIA card ***\n");
```

```
   printf("\tSamples the (16-bit) input from the incremental encoder\n");
   printf("\tby manipulation of the port C output lines.\n");
   printf("\t         v 1.1: (c) Feb 1994; Kees Lemmens\n\n");

   x=argc;
   while(--x>0)                    /* parse options */
   {   if(*argv[1]=='-')
       {   switch(*(++argv[1]))
           {  case 'n': count=atol(strchr(argv[1],'=') + 1);
                        if(count > UINT_MAX)
                        {   printf("Too many samples (max %u) !\n",UINT_MAX);
                            exit(1);
                        }
                        break;
              case 't': interval=atol(strchr(argv[1],'=') + 1);
                        break;
              default:  usage();
                        break;
           }
           ++argv; --argc;
       }
   }

   if(argc < 2)
       usage();

   if((out = fopen(argv[1],"w")) == NULL)
   {   printf("Error opening outputfile !\n");
       exit(1);
   }

   if((buffer = (Sample *) malloc((size_t)count * sizeof(Sample))) == NULL)
   {   printf("Failed to allocate buffer memory !\n");
       exit(1);
   }

#if (DEBUGLEVEL == 0)
   select_port_ac();
   set_portc_up(A0 | WR | RD | CS);
#endif

   printf("Estimated time required for this run : %ld sec.\n",
       (count * interval) / 1000);
   printf("You may now press the RESET button if you want ...\n");
   getchar();

   if(interval < MIN_REPORT)
       printf("Warning: interval too short to show runtime progress !\n");
```

```
   starttime = clock();
   for(x=0;x < count; x++)
   {
       read_porta(&buffer[x]);

       if(interval >= MIN_REPORT)       /* need enough time to report */
       {
#if (DEBUGLEVEL > 1)
           time_t tim=time(tim);
           printf("System time: %.24s ; ",ctime(&tim));
#endif
           printf("Time %6ld (ms); Value %5d \r",
               (long)(buffer[x].time / factor), buffer[x].value);

       }
       waitinterval(buffer[x].time + interval * factor);
       buffer[x].time -= starttime;
   }

   printf("\n");

   for(x=0;x < count; x++)
       fprintf(out,"%.1lf %d\n",buffer[x].time/factor,buffer[x].value);

   fclose(out);
   exit(0);
}
```

5.4 tandem program

As already mentioned, the oscillator is going to be modified in such a way that it will be possible to use two degrees of freedom instead of one.
Most likely is that there will be a second oscillator at the place where at this moment the windtunnel profile is mounted.
Although the setup itself isn't modified yet, the soft- and hardware for reading 2 cards simultaneously is already finished and seems to work without problems.
To registrate the position of both oscillators in realtime it is necessary to have a second incremental encoder, separate electronics and an extra interface card in the computer.
The software had to be changed slightly to make it possible to read and control 2 interface cards now.
A few remarks about the "tandem" :

- There was one tricky thing: the computer had to be fast enough to read and control the two cards in the way as described with the **readuid** program in the same 55 ms interval.

  This proved however not to be a serious problem for the 12 MHz AT computer used.

- The two interface cards couldn't be jumpered at the same memory location: it was necessary to give them both separate addresses.

  In this case $100 and $300 where chosen, so the software expects the cards to be at these locations, but it is sufficient to change the 2 defines in the `tandem.h` headerfile and recompile if it is necessary to use different base adresses.

- The moment of inertia for the second oscillator will certainly be much smaller than the moment of inertia for the first one.

  For this reason it was decided to use a frictionless type of incremental encoder: the ERO 1421. (also by Heidehahn)

  This encoder is electronically identical with the ROD426, but it doesn't have a ball bearing.

- The output format is not identical with that of the previous programs, as there are now two values for each timestamp:

  ```
  typedef struct _sample { clock_t time; int val1; int val2; } Sample;
  ```

  This implies the data needs some postprocessing with a spreadsheet or i.e. the GNUPLOT graphics program before it can be analysed.

This is the headerfile used for the `tandem` program. It is almost identical with the `uia.h` headerfile, except for the base addresses.

```
#define PPI_A       (0x0)
#define PPI_C       (0x2)
#define PPI_CONTROL (0x3)

#define PIT_RWCOUNT1 (0x4)
#define PIT_RWCOUNT2 (0x5)
#define PIT_RWCOUNT3 (0x6)
#define PIT_CONTROL  (0x7)

#define AD_C         (0x8)

/* next are registers in the THCT 2000 chip, that are controlled
   through the 4 highest bits of PORT C on the UIA card.
*/

#define A0 0x10
#define WR 0x20
#define CS 0x40
#define RD 0x80

#define READY 0x01

/* Base addresses for interface cards */
```

```
#define CARD1 0x100
#define CARD2 0x300
```

Next is a complete listing of the `tandem` program:

```
/* Program to read UIA Digital 8 bit input (port A) from a PC
   It also outputs control signals for the THCT 2000 encoder chip
   This version (TANDEM) reads 2 interface cards at the same time.
   Kees Lemmens; April. 1994.
*/

#define DEBUGLEVEL 0

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <math.h>
#include <limits.h>

#if (DEBUGLEVEL == 0)
#include <dos.h>
#endif

#include "tandem.h"

#define SAMPLES 100
#define DEFTIME 100

#define MIN_REPORT 500

typedef struct _sample { clock_t time; int val1; int val2; } Sample;

void select_port_ac(int card)
{
#if (DEBUGLEVEL == 0)
    outportb(card+PPI_CONTROL,0x91);   /* 1001 0001 naar control */
#endif
}

#if (DEBUGLEVEL == 0)
void set_portc_up(int card,int n)
{
   outportb(card+PPI_C,inportb(card+PPI_C) | n);
}

void clr_portc_up(int card,int n)
{
   outportb(card+PPI_C,inportb(card+PPI_C) & ~n);
```

```
}
#endif

void wait_for_ready(int card)
{
   /* wait for data to be available (READY signal goes low) */
   while(inportb(card+PPI_C) & READY);
}

void read_cards(Sample *s)
{
       s->time = clock();

#if (DEBUGLEVEL > 0)
      s->val1 = (int)2048*sin((double)s->time/200);
      s->val2 = (int)2048*cos((double)s->time/200);
#else
      clr_portc_up(CARD1,CS|RD);  /* pull CS & RD low, WR high (THCT2000) */
      clr_portc_up(CARD2,CS|RD);
      set_portc_up(CARD1,A0);
      set_portc_up(CARD2,A0);

      wait_for_ready(CARD1);
      s->val1 = (signed)inportb(CARD1+PPI_A);       /* Low byte */
      wait_for_ready(CARD2);
      s->val2 = (signed)inportb(CARD2+PPI_A);       /* Low byte */

      clr_portc_up(CARD1,A0);
      clr_portc_up(CARD2,A0);
      wait_for_ready(CARD1);
      s->val1 += (signed)inportb(CARD1+PPI_A) * 256;  /* High byte */
      wait_for_ready(CARD2);
      s->val2 += (signed)inportb(CARD2+PPI_A) * 256;  /* High byte */

      set_portc_up(CARD1,CS|RD|WR|A0);  /* CS & RD & A0 & WR high */
      set_portc_up(CARD2,CS|RD|WR|A0);
#endif
}

void waitinterval(clock_t newtime)
{   clock_t curtime = clock();

#if (DEBUGLEVEL > 1)
    printf("newtime %ld currenttime %ld\n",newtime,curtime);
#endif

    if(curtime >= newtime)
    {    printf("Interval too short or system too slow !\n");
```

```
        exit(1);
    }
    while(clock() < newtime);
}

void usage(void)
{  printf("Usage: tandem [-n=<nr>] [-t=<nr>] <file>\n\n");
   printf("  -n   Number of samples (default %u)\n",SAMPLES);
   printf("  -t   Approx. time between 2 samples (default %ld ms)\n",DEFTIME);
   printf("  file Send output to <file> (con: sends to screen)\n");

   exit(1);
}

void main(int argc,char *argv[])
{
   long count=SAMPLES;
   unsigned x;
   Sample *buffer;
   clock_t starttime; /* to start at zero */
   clock_t interval=DEFTIME;
   FILE *out;
   double factor = CLK_TCK / 1000.0;

   printf("\t*** TANDEM: program to read port A from 2 UIA cards ***\n");
   printf("\tSamples the (16-bit) input from the incremental encoders\n");
   printf("\tby manipulation of the port C output lines.\n");
   printf("\t         v 1.0: (c) Apr. 1994; Kees Lemmens\n\n");

   x=argc;
   while(--x>0)                    /* parse options */
   {   if(*argv[1]=='-')
       {   switch(*(++argv[1]))
           {  case 'n': count=atol(strchr(argv[1],'=') + 1);
                        if(count > UINT_MAX)
                        {   printf("Too many samples (max %u) !\n",UINT_MAX);
                            exit(1);
                        }
                        break;
              case 't': interval=atol(strchr(argv[1],'=') + 1);
                        break;
              default:  usage();
                        break;
           }
           ++argv; --argc;
       }
   }
```

```
   if(argc < 2)
       usage();

   if((out = fopen(argv[1],"w")) == NULL)
   {   printf("Error opening outputfile !\n");
       exit(1);
   }

   if((buffer = (Sample *) malloc((size_t)count * sizeof(Sample))) == NULL)
   {   printf("Failed to allocate buffer memory !\n");
       exit(1);
   }

#if (DEBUGLEVEL == 0)
   select_port_ac(CARD1);
   select_port_ac(CARD2);
   set_portc_up(CARD1,A0 | WR | RD | CS);
   set_portc_up(CARD2,A0 | WR | RD | CS);
#endif

   printf("Estimated time required for this run : %ld sec.\n",
       (count * interval) / 1000);
   printf("You may now press the RESET button if you want ...\n");
   getchar();

   if(interval < MIN_REPORT)
       printf("Warning: interval too short to show runtime progress !\n");

   starttime = clock();
   for(x=0;x < count; x++)
   {
       read_cards(&buffer[x]);

       if(interval >= MIN_REPORT)      /* need enough time to report */
       {
#if (DEBUGLEVEL > 1)
           time_t tim=time(tim);
           printf("System time: %.24s ; ",ctime(&tim));
#endif
           printf("Time %6.0lf (ms); Value 1 %5d; Value 2 %5d \r",
             (buffer[x].time / factor), buffer[x].val1, buffer[x].val2);
       }
       waitinterval(buffer[x].time + interval * factor);
       buffer[x].time -= starttime;
   }

   printf("\n");
```

```
    for(x=0;x < count; x++)
        fprintf(out,"%.1lf %d %d\n",buffer[x].time/factor,
          buffer[x].val1,buffer[x].val2);

    fclose(out);
    exit(0);
}
```

5.5 Some other utilities

To facilitate processing of data acquired through IPCOACH or **readuid**, a few small utilities where written that will be described very shortly hereafter:

- ipc2asc

```
        *** IPC2ASC: Convert binary IPCOACH 3.1 files to ASCII ***
                  v 1.3: (c) Dec 1993; Kees Lemmens

Usage: ipc2asc [-setnr=<nr>] <inputfile> [<outputfile>]
```

  The output from the IPCOACH program is always a binary file. As it is much more convenient to use ASCII files with i.e. the GNUPLOT program that was used to make the graphics in this report, it was decided to build a small conversion program to convert binary IPCOACH 3.1 output (.dt0 files) to ASCII.

  This program only works with IPCOACH 3.1 data files (with 4000 samples per dataset) and by default it will only extract the first set of data if there is more than one set in the file.

  To extract other sets from the binary file the '-s' switch can be used. Setnr 0 extracts the first set, setnr 3 will extract number 4.

  The output will be sent to stdout by default, so under UNIX pipes and redirection can be used. Under DOS this can be useful with the **more** pager.

  The output will contain 2 numbers for every sample: a timestamp and a corresponding value.

- readuia

```
        *** READUIA: program to read AD converter ***
               v 1.1: (c) Dec 1993; Kees Lemmens

Usage: readuia [-c=<nr>] [-n=<nr>] [-t=<nr>] <file>

  -c   Supply channelnumber (default 1)
  -n   Number of samples (default 100)
  -t   Approx. time between 2 samples (default 100 ms)
  file Send output to <file> (con: sends to screen)
```

This program reads the UIA/UIB Analogue Digital Converter and sends the output in ASCII to the output file.

The output file contains two numbers for every sample : a timestamp and the ADC value.

It can be used instead of IPCOACH for simple straightforward measurements. The advantage is that it isn't restricted to 4000 samples, as is the case with IPCOACH 3.1.

The channelnumbers correspond with the channels on the CMA measurement panel.

- uiaplot

```
        *** UIAPLOT: Simple program to plot UIA data files ***
                 v 1.1: (c) Dec 1993; K. Lemmens

Usage: uiaplot [-xsc=<nr>] [-ysc=<nr>] [-hoff=<nr>]
               [-voff=<nr>] <inputfile>
```

This program can plot ASCII datafiles either made by ipc2asc, readuia, the **readuid** program or even with an editor.

It expects every plot to contain 2 values: a timestamp and a sample value. The timestamp will be used as X-coordinate; the sample value will the Y-coordinate.

The flags can be used to rescale or offset the plot. The program doesn't do any clipping, so it is important to check if the plot fits on the screen with the applied flags.

If no flag is supplied it will auto determine the scale factors to put the plot on the screen as large as possible.

- uiaconv

```
        *** UIACONV: Converts ASCII data files (i.e. UIA output) ***
                 v 1.3: (c) Dec 1993; Kees Lemmens

Usage: uiaconv [-xsc=<nr>] [-ysc=<nr>] [-hoff=<nr>] [-voff=<nr>]
               <inputfile> [<outputfile>]
```

This program can be used to rescale and/or offset x/y coordinates in ASCII sample files.

Output is sent to stdout by default. (same as ipc2asc)

The uiaconv, uiaplot and ipc2asc programs do not require any special hardware, so they can also be recompiled with a standard ANSI C-compiler under UNIX, VMS or whatever other OS.

Chapter 6

Conclusions

Some conclusions of this research may be summarised as follows:

- The analogue sensor as used in the first experiments (the variable density filter) was an inaccurate sensor. A digital sensor as described in this report gives much better and more accurate results.
- Although digital sensors are often more complicated than analogue ones, they can be very accurate.

  Moreover: while the accuracy of an analogue encoder depends on a number of almost unpredictable variables, the accuracy of a digital encoder can be predicted very exactly and in an easy way.
- It became obvious that software is a much more easy way to control these sensors than trough hardware: hardware modifications require at least a few weldings and often many extra components (with a chance of damaging the setup, especially after many modifications !), while software can be modified quickly with a simple editor.

  And: if the software doesn't do what it is supposed to do, it is enough to move the backup (provided there is one) over the new version and all modifications are cleared.

  This is exactly the reason why during development more and more functionality was moved from the hardware to the software.
- It is comparatively easy to output the data to a datafile with a digital sensor: for the analogue sensors always an Analogue Digital Converter that will result in a further decrease of accuracy is necessary, while digital output can be processed directly by every computer without any extra loss off precision.

Appendices

.1 List of acquired data

The following is a list of all datafiles acquired while developing the Realtime Postition Determination Mechanism. The time field in this list shows the duration of the measurement.

Filename	Time	Description
1811os1.out	5 s	original variable density filter
1811os2.out	5 s	adapted variable density filter
1811os3.out	200 s	irregularities in variable density filter
0112os1.out	200 s	adapted variable density filter
0112os2.out	200 s	0112os1.out continued
0112li1.out	100 s	var. den. filter: lights periodically switched on/off
0312cmp1.dt0	20 s	var. den. filter + static sensor (2 measurements)
0312cmp2.dt0	200 s	var. den. filter + static sensor (2 measurements)
0312cm2a.out	200 s	0312cmp2.dt0 : var. den. filter
0312cm2b.out	200 s	0312cmp2.dt0 : static sensor
0312dif1.out	200 s	subtract 0312cm2b.out from 0312cm2a.out
0312dif2.out	200 s	subtract (2 * 0312cm2b.out) from 0312cm2a.out
0312de1.out	600 s	var. den. filter: examine system damping
0312li2.out	100 s	var. den. filter: lights periodically switched on/off
1712h2o1.out	10 s	water potentiometer: long bar on side
1712h2o2.out	10 s	water pot.meter: short pin in middle, offset 50Hz buzz
1712h2o3.out	10 s	water pot.meter: short pin in middle
1712h2o4.out	600 s	water pot.meter: examine system damping
3112prgc.out	180 s	first result reading UIA AD converter from C program
2101enc1.out	10 s	digital incremental encoder: LED output signal 1
2101enc2.out	0.1 s	digital incremental encoder: LED output signal 2
1802enc3.out	6 s	output digital encoder interface and software 1
1802enc4.out	6 s	output dig. enc. interface 2
1802enc5.out	350 s	output dig. enc. interface 3: examine system damping
1802enc6.out	35 s	output dig. enc. interface 4

Most of these datafiles are in this report in the form of a graph. A few of them were dropped because they resembled another set very close.

Bibliography

[1] R. D. Blevins , *Flow induced vibration*, Van Nostrand Reinhold, New York, 1990.

[2] T. I. Haaker & A. H. P. van der Burgh, *On the dynamics of aeroelastic oscillators with one degree of freedom*, Report TWI 92-96, Faculty of Technical Mathematics and Informatics, Delft University of Technology, 1992.

[3] B. W. van Oudheusden, *Investigation of an Aeroelastic Oscillator : Analysis of One-Degree-of-Freedom Galloping with combined Translational and Torsional Effects*, Report LR-707, Faculty of Aerospace Engineering, Delft University of Technology, 1992.

[4] B. W. van Oudheusden, *Development of an aeroelastic oscillator : design and initial results of an experimental set-up*, Report LR-746, Faculty of Aerospace Engineering, Delft University of Technology, 1993.

[5] W. E. Boyce and R. C. DiPrima, *Elementary Differential Equations and Boundary Value Problems*, (4th ed.), (New York: John Wiley & Sons, 1986).

[6] Alonso & Finn, *Fundamentele Natuurkunde deel 1: Mechanica*, (Overberg: Delta Press, 1989).

[7] Didaktiek Natuurkunde, *Fysische Informatica deel 2*, (Amsterdam: Universiteit van Amsterdam, 1990).