Extracting more accurate position and velocity estimations using time stamping

T.A.C. Verschuren

DCT 2006.58

Bachelor's Thesis

Dr. Ir. M.J.G. van de Molengraft (supervisor)
Ir. R.J.E. Merry (coach)

Eindhoven University of Technology
Department of Mechanical Engineering
Control Systems Technology Group

Eindhoven, June 23, 2006

Abstract

Angular position and velocity information are often required in control systems design of mechanical machines, because control is applied by using an angular measurements of the position or velocity. Nowadays, incremental optical encoders are widely used for measurement purposes since the majority of precision motion devices are driven by rotary motors. The measurement accuracy, which is determined by the encoder resolution, limits the maximum achievable performance. Improving the accuracy by means of estimations can prevent the need for more accurate, more expensive measurement devices.

In this thesis, a method, called time stamping, is proposed to increase the accuracy of encoder measurements. Time stamping is based on collecting time instants at which quadrature events take place. Polynomial fitting through previous collected time stamps and extrapolating this fit provides an accurate position estimation. Also, a more accurate angular velocity can be determined.

Simulations on this matter show that the position estimation can be radically improved to errors of only hundredths of an encoder increment. Furthermore, dependency of several parameters such as angular velocity, disturbances and number of time stamps is investigated. A 2^{nd} order polynomial fit calculated from 3 time stamps performed best.

The concept is implemented on a experimental set-up, consisting of an DC motor and a TUeDACS/1 MicroGiant to log quadrature events. The time stamping method is compared to the conventional way of estimating the position. Although optimization of the implementation is needed, position measurements for constant velocity profiles can be improved with approximately 70 % at a sampling frequency of approximately 100 Hz

Contents

Chapter 1

Introduction

1.1 Motivation

Angular position and velocity information is often required in motion control systems design of mechanical machines, because control is applied by using an angular measurement of the position or velocity. Nowadays, incremental optical encoders, also known as shaft encoders, are widely used for this purpose since the majority of precision motion devices are driven by rotary motors. The maximum accuracy of the controlled system is limited by the measurement accuracy. When measuring position by using incremental optical encoders with limited resolution, the occurrence of quantization errors is inevitable. Differentiating encoder measurements to obtain velocity information results in an amplification of the quantization errors. The performance can be increased by improving the measurement accuracy, i.e. using a higher resolution encoder. However, this is an expensive option. Previous and current research focusses on finding alternative solutions to improve the accuracy.

1.2 Problem definition

A possible solution is the concept of time stamping with extrapolation. The time stamping concept is implemented using a low resolution encoder with a TUeDACS/1 MicroGiant [14]. It improves the accuracy of the position and velocity estimation, using only the encoder measurements. The MicroGiant has an encoder time stamp register, which stores counter values of the quadrature signal produced by the encoder and the time instant at which the corresponding transition has occurred. The combination of a time instant and a counter value is called a time stamp. Time stamps from previous transitions can be used to achieve a more accurate position estimation than the position measurement of the encoder by fitting a polynomial through these points. Furthermore, it is possible to prevent time stamps from entering the register. This can be done by skipping a number of transitions of the quadrature signal or by introducing a delay. In this way, more history of the position is taken into account as is done by storing all transitions. These additional options may lead to a better position and velocity estimation. The subject of this project is to investigate the time stamping concept, skip and delay options, and finally implement it by using a MicroGiant. In order to determine the behaviour of the time stamping concept answers are sought to the following questions:

- What is the influence of using different kinds of polynomial fits? One can think of using different order of fits or taking into account a different number of time stamps.

- What is the influence for different velocity profiles?
- Under which circumstances should one use the skip option, the time delay option or neither of them?

1.3 Thesis layout

This thesis basically follows the chronological order of the project. Therefore it has three main phases:

- *Initiation or exploration phase*, during which literature study was carried out to get familiar with incremental optical encoders, the concept of time stamping, possibilities of the TUeDACS/1 MicroGiant and previous research done in this field (see Chapter 2 and 3).
- *Simulation phase*, during which simulations were done to investigate the time stamping concept and determine the influence of different parameters. Descriptions and results of the simulations can be found in Chapter 4.
- *Experimental phase*, during which the time stamp concept will be implemented on an experimental set-up (see Chapter 5).

Finally, conclusions are drawn and some recommendations are given. These will be discussed in Chapter 6.

Chapter 2

Overview of the literature search

In the initiation phase of the project, a research of the existing literature on position and velocity estimation has been carried out. Purpose of this research was to get familiar with incremental optical encoders, the concept of time stamping and to collect different kind of approaches to improve angular position and velocity estimations. In this chapter, an overview of the literature will be presented.

2.1 Keywords for literature search

- Incremental optical encoder, shaft encoder
- Position-velocity estimation or improvement
- Polynomial fitting, interpolation, fit algorithms
- Fixed-time, fixed-position
- Quantization errors

2.2 Functional overview of incremental optical encoders

The incremental optical encoder consists of three basic components: a slotted disk, a light source and a dual light-detector. As shown in Fig. 2.1. The light source shines on a disk which is covered with a regularly-spaced radial pattern of transmissive and reflective/absorptive elements called encoder increments. Low resolution encoders use a disk with radially-cut slots to vary optical transmissivity of the disk. High resolution encoders typically use a radial pattern of chromium lines evaporated onto clear glass. A dual light detector on the opposite side of the disk detects varying intensities of light as the disk rotates due to the pattern on the disk.

The output of the light detectors are two signals, (a, b) in Fig. 2.1, which are correlated with the intensity of the light striking each of the two detectors. Ideally, the signals are sinusoidal waveforms. The two signals are in quadrature, i.e. 90° phase shift with respect to each other. This makes it possible to determine the direction of motion by detecting the leading signal. In digital encoders, the two detector outputs are converted into digital quadrature signals using thresholding circuits. Due to the quadrature phase between the two

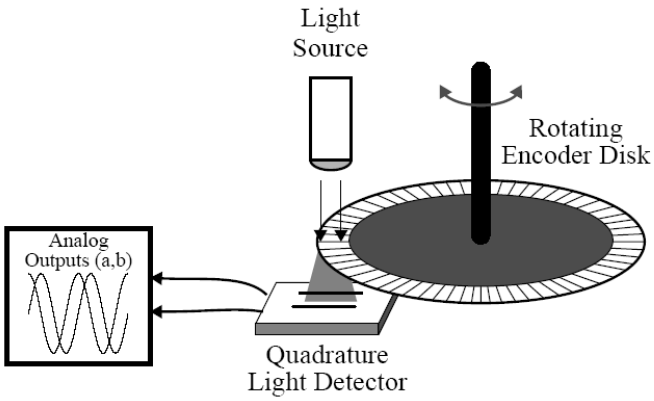
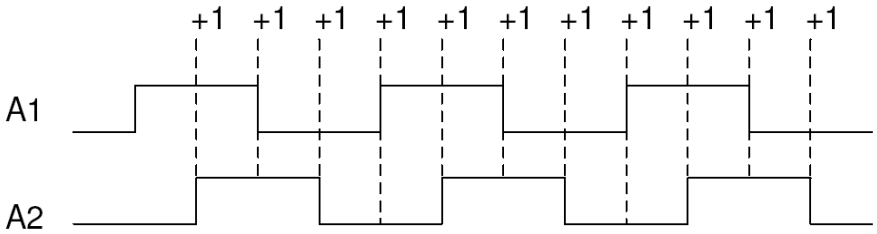


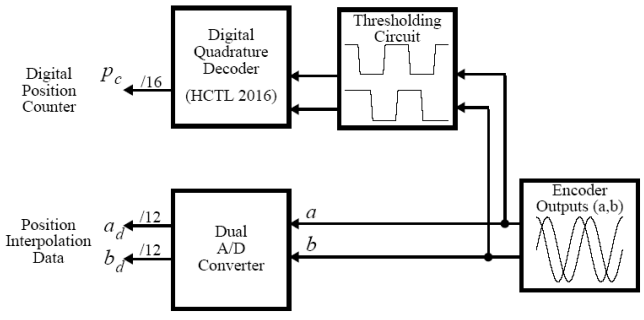
Figure 2.1: Functional principle of an encoder

signals, position changes of 0.25 of a line can be converted into a pulse train signal. A digital counter counts "up" and "down" changes and uses this counter value as an estimate of the encoder position. This can be seen in Fig. 2.2(a). Most incremental encoders in use today are digital encoders.

An analog encoder does not convert the signals in digital quadrature signals. Assuming that the sinusoidal waveforms are symmetric around 0 volts, the zero-crossings of the two signals can be used to determine the rotary position. The treatment of the signals is shown in Fig. 2.2(b).



(a) Converting to a pulsetrain



(b) Treatment of the signals

Figure 2.2: Encoder signals

2.3 Overview of relevant publications

The purpose of the literature research was to get an overview off the available approaches to improve angular position and velocity estimations. In previous research many methods are presented. In the following, the most important ones will be discussed.

A first method to get more accurate angular position and velocity estimation is to use a Kalman filter. This was done for the first time by L. Ljung and T. Glad in 1984 [1]. The problem can be seen as a state estimation problem. The equations of the electrical motor can be represented in the state-variable form where the state of the system contains among other things the position and velocity of the encoder shaft. The Kalman filter estimates the position and the velocity based on noisy observations. This means that system noise and measurement noise is taken into account and so the problem is stochastic. The Kalman filter is a recursive estimator. This means that the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current state. At each discrete time increment a new state can be generated with the following stochastic equations:

$$\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k + \mathbf{w}_k \tag{2.1}$$

$$\mathbf{z}_k = H\mathbf{x}_k + \mathbf{v}_k, \tag{2.2}$$

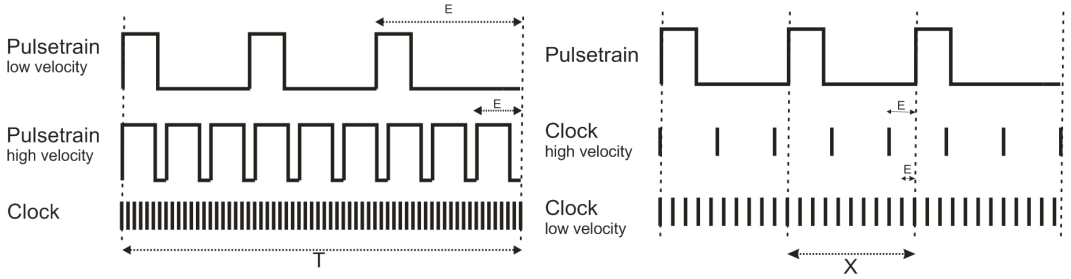
where the column vector $\mathbf{x}_k$ is the state of the system, $\mathbf{z}_k$ is an observation or measurement, $\mathbf{w}_k$ is Gaussian system white noise with covariance Q and $\mathbf{v}_k$ is Gaussian measurement white noise with covariance R.

The Kalman filtering method is also used in similar approaches. For instance in the approach that P.R. Belanger presented in 1992 [2]. In addition to the Kalman filter this method uses a model of the angle signal. The approach of Y. Buchnik and R. Rabinovici presented in 2004 [3] proposes a method to detect and control the position and velocity of a motor with a low resolution encoder at very low speeds of several rad/s. The low resolution of the encoder is equivalent to a quantization error in the position estimation. Furthermore, the presence of a unknown load torque is equivalent to a large system noise. The Kalman filter can be used to estimate this unknown load torque. The inputs of the Kalman filter are the motor current and the previous estimated position. The outputs are the estimated position, velocity and load torque.

To estimate the velocity of the encoder shaft two classic approaches, called the fixed time approach and the fixed position approach, exist. From these velocity estimations, one can construct a position estimation by integrating. The most common approach is the fixed time method. The fixed time method measures the distance traveled over a fixed time period. A discrete time system with a particular sampling period can measure how far the encoder has turned by counting the lines registered during that period. The fixed position method measures the time required for the encoder to travel a fixed distance. The time required can be determined by counting the clock pulses of data acquisition between a fixed number of encoder lines. In the past few years, a lot of velocity estimation algorithms based on the two principles described above have been developed, i.e. in 1992, by R.H. Brown, S.C. Schneider and M.G. Mulligan [4], in 1995 by the same authors [5] and in 1996 by M.F. Benkhoris and M.A. Ahmed [6] some of those algorithms are presented. In the latter both fixed time and fixed position algorithms are discussed. The simplest fixed time velocity estimation is the lines per period (LPP) estimator, also called the finite difference method (FDM). If $\Delta x_k = x_k$ - x_{k-1} is the number of encoder increments counted and the time span of an interval is

T, then the velocity can be determined as $v_k = \Delta x_k / T$. The simplest velocity estimation using the fixed position approach is the reciprocal time or inverse time method. The velocity is defined as $v_k = 1/\Delta t_k$ when the distance between time measurements is one increment and $\Delta t_k = t_k$ - t_{k-1} is the time to travel that distance. Furthermore algorithms like the average filter (AVE), backward difference expansion (BDE), Taylor series expansion (TSE) and least-squares fitting (LSF) are presented [5].

The fixed time approach and the fixed position approach have a big disadvantage. Both approaches asymptotically break down at certain circumstances. The fixed time methods break down at low velocities (respectively high sampling rates) due to the fact that the noisy component of the position information gets more and more influence. The fixed position methods gets inaccurate at high velocities (respectively low sampling rates). So both methods ensure good performance only in a restricted range of velocities.



(a) Inaccuracy at fixed time

(b) Inaccuracy at fixed position

Figure 2.3:

In 2003, S. D'Arco, L. Piegari and R. Rizzo dealt with this problem in [7]. In this paper an innovative hybrid algorithm is proposed. The objective was to guarantee a reduced relative error in a wide range of velocities. The hybrid algorithm combines the basic ideas of both fixed time and fixed position methods. Based on the errors that occur in the two methods, a complex formula to calculate the velocity is determined. A constant sampling frequency, encoder resolution and measuring time are assumed. Now the estimated velocity is only a function of the real velocity and the initial position of the encoder. An advantage of this approach is a better accuracy of the estimated velocity at a wide range of speeds and the algorithm is quite easy to implement.

Another approach which partly deals with the problem of break down is presented by F. Janabi-Sharifi, V. Hayward and C.J. Chen in 2000 [8]. The average filter in [5] uses averaging of the LPP estimates over a number of periods. This is a common practice to decrease the noise effect, but if the number of periods is too large, the estimated velocity would have indolence to follow large changes in velocity at once. In [8], the authors propose a method called first-order adaptive windowing (FOAW) which deals with the number of periods. In order to trade noise reduction against control delay and precision against reliability, the number of periods (the window size) should be selected adaptively depending on the measured signal. The windows size should be short when the velocity is high, yielding more reliable estimates and faster calculations; it should be large when the velocity is low to produce more precise estimates. A first-order adaptive window algorithm is presented in which the size of the window is determined by the slope of the linear approximation between two samples x_k and x_{k-m}. At low velocities FOAW resembles the inverse time method and at high velocities it resembles

the finite difference method. The computational costs however are a big disadvantage of the FOAW approach.

A second non-conventional approach for increasing the position and velocity information is to interpolate the sinusoidal analog encoder outputs. This idea is discussed in [9] by N.C. Cheung in 1999 and in [10], [11] and [12] by K.K. Tan respectively in 2002, 2004 and 2005. The basis idea of the interpolation method is to derive high-order sinusoids based on the fundamental one. This results in more zero crossings for each increment and thus in a higher resolution. Ideally, the quadrature encoder signals are identical sinusoidal signals displaced by a 90° phase shift. Given the values of $\sin(a)$ and $\cos(a)$, $\sin(2a)$ and $\cos(2a)$ can be derived using $\sin(2a) = 2 \cdot \sin(a) \cdot \cos(a)$ and $\cos(2a) = 1 - 2 \cdot \sin(a)^2$. This can also be done for $\sin(na)$ and $\cos(na)$. An analog comparator may be used to transform the high-order sinusoids into a pulse train. A second, more efficient way, is to do this within a look-up table. Then $\sin(na)$ and $\cos(na)$ entries can be directly converted into binary values.

2.4 Overall remarks

As can be seen from the above literature search, there are a lot of approaches to extract more accurate position and velocity estimation from the quadrature signals of the incremental optical encoder. To get a clear overview, categorization of the possible methods can be helpful.

A first step can be made when one looks at the way the encoder output is treated. Does the approach use the quantized signals or the analog ones? Most approaches, like the Kalman filtering method, fixed time and fixed position method, use the quantized signals. The interpolation technique proposed by among others K.K. Tan in [11] uses the analog outputs.

A further distinction between the approaches could be made in the way the position and/or velocity estimation is calculated. Is the approach model-based or does it use algorithms like the finite difference method or inverse time method and their extractions? The Kalman filtering method in [1],[2] and [3] uses dynamical equations as a model for the system. The hybrid algorithm in [7] and the adaptive windowing technique in [8] are based on algorithms.

Although every method that's described here has advantages, none of them is perfect. The fixed time or fixed position based algorithms get inaccurate at low and high angular velocity, respectively. The approaches which use the analog outputs of the encoder are also not suitable because TUeDACS/1 devices generate quantized signals. When using a Kalman filter to estimate position and velocity, a model of the system is necessary. To avoid these limitations a new approach has been developed, called "time stamping". In this approach, the angular position will be estimated by extrapolating a polynomial fit based on quadrature counts and their corresponding time instants. Angular velocity can be determined by differentiation of this polynomial. Fig. 2.4 shows an schematic overview of all approaches.

The time stamping approach that's investigated during this project can be seen as a fixed position approach and shows similarities to the least-squares velocity estimation approach of [6]. However, there is a large difference, the first-order adaptive windowing estimates only the velocity, but the time stamping approach estimates the position. Both approaches do take into account samples that occurred earlier in time. The number of samples is variable. In the following chapter, the concept of time stamping will be discussed.

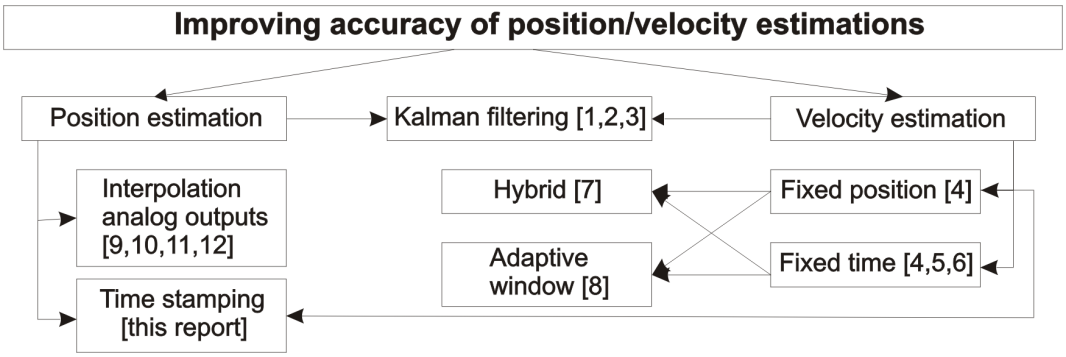


Figure 2.4: Overview of all approaches

Chapter 3

Concept of time stamping

To control an electrical motor, information of its position and/or velocity is needed to obtain a good tracking performance. The common way to determine the position is to read out the encoder counter value at a certain sampling frequency of the controller. The number of encoder increments registered by the counter is a measure for the distance the motor has traveled. The measured signal is quantized because of the finite resolution of the encoder. Therefore, a quantization error on the position, with a maximum value of half an encoder step, is introduced. To obtain the angular velocity, the quantized position signal can be differentiated. However, the discontinuities in the position signal will result in spikes in the velocity signal.

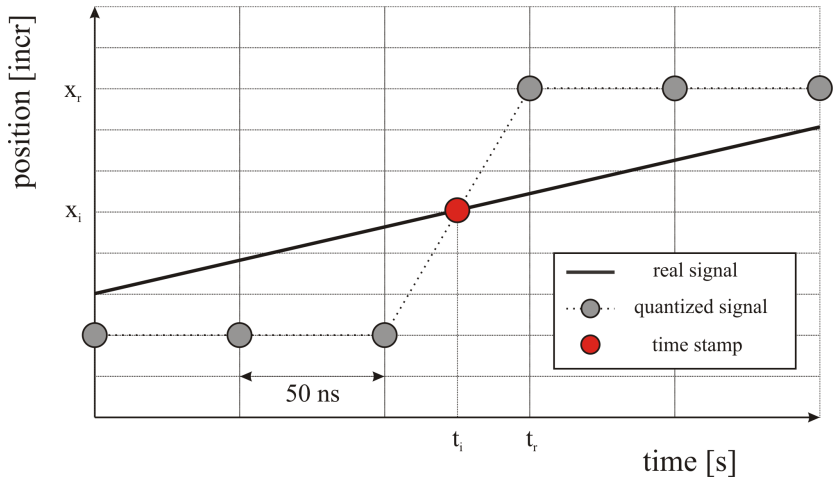


Figure 3.1: Concept of time stamping

3.1 Definition time stamps

To extract a more accurate position and a more useful velocity signal from the encoder, the idea of time stamping is introduced. Time stamping is also known as the event detection method. An event refers to the transition of one encoder increment to the next. At such step, the position x_i is exactly known, namely halfway the two encoder steps (See Fig. 3.1).

It is possible to detect the time instant t_i at which such a quadrature event takes place. The combination (x_i, t_i) of a time instant t_i and the matching position x_i is called a time stamp. The accuracy by which quadrature events can be detected is dependent on the clock frequency $f_c = 1/T_c$. In this project the MicroGiant is used. In this device time stamps are evaluated with a resolution of 50 ns or with a frequency of 20 MHz. The MicroGiant registers a quadrature event at time instant t_d, so the matching position would be x_d and the time stamp would be (x_d, t_d). Since, the resolution α of the encoder is known the actual corresponding position equals $x_i = x_d/4\alpha$. When encoder imperfections are neglected, the position therefore can be corrected to x_i. The time stamp resolution of 50 ns is assumed small enough to neglect the difference between t_i and t_d. Finally, a time stamp consists of the pair (x_i, t_d). By using the time stamps of a certain amount of previous events, a position estimation can be obtained by fitting a polynomial through these points. This fit can be extrapolated to have a very accurate prediction of the position at the time instant determined by the sampling frequency of the controller.

3.2 Extrapolation

In the concept of time stamping, extrapolation plays a significant role. The order of polynomial fitting and the number of time stamps that are used to calculate the fit are important parameters. There are several methods to fit a polynomial through a discrete set of points. One of the most computationally convenient ways is the method of least squares [15], which can be presented as solving a set of linear equations:

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{3.1}$$

where $\mathbf{A} = \begin{bmatrix} 1 & \cdots & t_1^{m-1} & t_1^m \\ 1 & \cdots & t_2^{m-1} & t_2^m \\ 1 & \cdots & \cdots & \cdots \\ 1 & \cdots & t_n^{m-1} & t_n^m \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} c_0 \\ \cdots \\ c_{m-1} \\ c_m \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix}$ with

- n number of time stamps
- m order of the polynomial fit
- c_i polynomial coefficients, with $i = 1, ..., m$
- t_j time instants, with $j = 0, ..., n$

Assuming $\mathbf{A}$ is an $m \times n$ matrix with $m > n$, $\mathbf{b}$ is a $n \times 1$ vector and $\mathbf{x}$ is a $m \times 1$ vector. The solution to this overdetermined system then can be formulated as follows:

$$\mathbf{x} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} \tag{3.2}$$

To solve this equation for $\mathbf{x}$ and minimize the computational costs, LU decomposition is used. The matrix $\mathbf{A}^T\mathbf{A}$ is decomposed into a product of a lower triangular matrix, $\mathbf{L}$, and an upper triangular matrix, $\mathbf{U}$. The linear system $\mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ can be written as $\mathbf{L}\mathbf{U}\mathbf{x} = \mathbf{A}^T\mathbf{b}$ and hence can be solved by first solving the lower triangular system $\mathbf{L}\mathbf{y} = \mathbf{b}$ by forward-substitution, followed by the upper triangular system $\mathbf{U}\mathbf{x} = \mathbf{y}$ by back-substitution. Now the

coefficients of the polynomial are known and an estimation of the position on a certain time instant can be determined with this polynomial,

$$x_f = c_m t^m + c_{m-1} t^{m-1} + \cdots + c_0. \tag{3.3}$$

From Fig. 3.2 it can be seen that the estimated position x_f is a better approximation to the real position than the quantized measurement x_q.

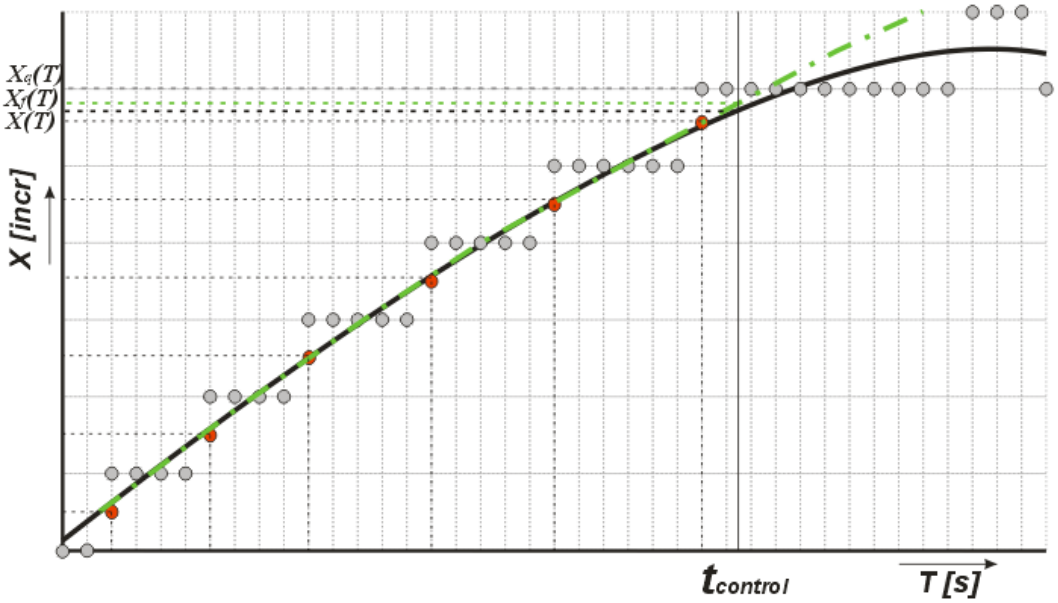


Figure 3.2: An example of a fit procedure

The concept of time stamping can be extended with additional options that are available on the MicroGiant. The MicroGiant contains a skip option and a time delay option.

The skip option involves the possibility to prevent a number of quadrature events s from storing into the register, but every $s+1$ change of the quadrature counter, with its including time stamp, is stored.

The time delay option involves the possibility to prevent time stamps from storing into the register during a delay time Δt, but the first quadrature event that occurs after Δt is stored.

More on these options will be discussed in Chapter 5.

Chapter 4

Simulations

The concept of time stamping is first investigated by means of simulations. The simulations are done for several reasons. The first purpose for doing simulations is to gain insight in the concept of time stamping and quantization. Furthermore, the benefits of time stamping over the common quantized measurements have to be investigated. Also the influence of the two additional options, the skip and the delay time option, are evaluated. Do these possibilities provide a better position estimation and in which cases? One could evaluate the benefits of time stamping in many different ways, since there are a lot of parameters that can be investigated. The position error depends on the encoder resolution, the input signal, the order of the fit, the number of time stamps taken into account, etc. Therefore, before running simulations several assumptions have to be made.

4.1 Assumptions

4.1.1 Encoder model

First thing to do is to determine what kind of encoder will be used during the simulations. What is the resolution of the encoder? Are there encoder imperfections? For all simulations an encoder with 100 increments will be used, since this is also the resolution of the encoder which will be used in the experiments. Due to the quadrature phase between the two signals, position changes of 0.25 of an increment can be converted into a pulse train signal, so the encoder resolution will be 400 $pulses/rev$. Furthermore the assumption is made that it is an ideal encoder, so there are no imperfections present.

4.1.2 Input signals

Incremental optical encoders are mainly used to control precision motion systems. Typical input signals for these systems are constant velocity with some sinusoidal disturbance. Transient behaviour is unlikely to occur. The input signals for the simulations will be of the form

$$x_{in} = vt + A\sin(2\pi ft), \tag{4.1}$$

where v is the nominal velocity, f the frequency of the disturbance, and the amplitude A determines the velocity deviation of the nominal value.

4.1.3 Definition of the error

To evaluate the accuracy of the position estimate and to compare results obtained by simulations, a definition for the error has to be made. In all simulations, the errors e_q and e_f will be defined as the difference between the input signal and respectively, the quantized signal and the polynomial fit at the first control sample taken in time. So,

$$e_q = x_{in} - x_q \tag{4.2}$$

$$e_f = x_{in} - x_f \tag{4.3}$$

These errors will be expressed in encoder increments. To quantify these errors the root mean square value will be determined over at least one period of the sinusoidal disturbance on the velocity signal.

4.2 Simulating the time stamping concept

4.2.1 A first procedure

As already mentioned before, the purpose of the simulations was to gain insight in the time stamping concept and to determine what the influence of the parameters is. In the simulations an artificial quantization is performed to resemble the real system. Next, an array with position information and an array with matching time instants are composed. A time stamp pair is added when a change in the quantized signal occurs. The position is reduced with half an encoder increment, because the number of encoder increments is known (see Fig. 3.1). The number of time stamp pairs that is stored into the arrays can be varied. Then the coefficients of the polynomial fit are calculated and the position is estimated. As a first test, simulations with 3 polynomial fits of different order are carried out to check how the time stamping concept works as a position estimator. The input signal that is used is a constant velocity with a sinusoidal disturbance. The nominal velocity was $v = 100\ rad/s$, the sinusoidal disturbance frequency was $f = 200\ Hz$ and the sinusoidal disturbance amplitude was 10% deviation on the nominal velocity, i.e. $A = 10\ rad/s$. Fig. 4.1 shows the real signal, the quantized signal, the time stamps and the estimations with the three different fit orders. The polynomial fits are based on 5 time stamps. Fig. 4.2 shows the position error over 1 period of the sinusoidal disturbance. It can be seen that the time stamping concept decreases the position error enormously. While the position error from a quantized signal can be up to 0.5 encoder increments, the errors from the polynomial fits of 2^{nd} and 3^{rd} are significantly reduced to only hundredths of encoder increments. The 1st order fit doesn't improve the position estimation very much because it is not able handle the curvatures of the sinusoidal disturbance. This disturbance can be seen very clearly in the position error with this polynomial fit.

4.2.2 Determining the influence of different parameters

To determine the influence of the parameters that are of interest in the time stamping concept additional simulations are carried out. The parameters that will be investigated are:

- Angular velocity

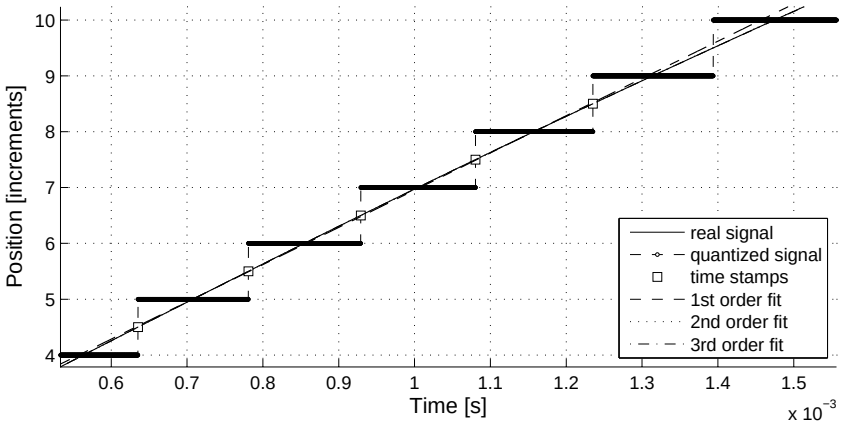


Figure 4.1: Position estimation with 3 different polynomial fits

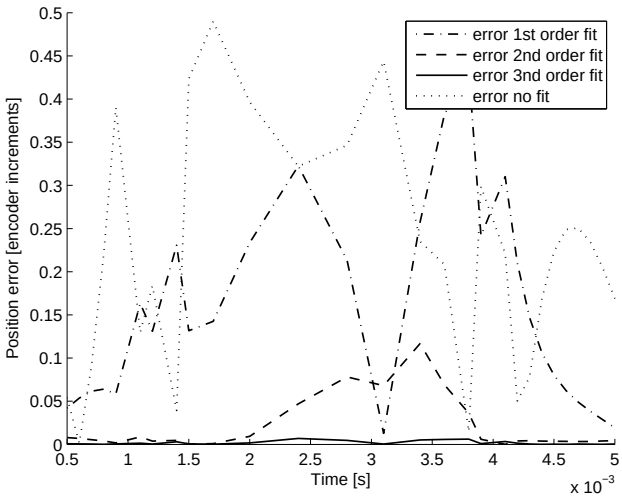


Figure 4.2: Position error over 1 period of the disturbance

- Disturbance frequency

- Fit algorithm parameters (number of time stamps, order of polynomial fit)

The TUeDACS/1 MicroGiant can store 5 time stamps. The polynomial fits of interest are $1^{st}, 2^{nd}$ and 3^{rd} order. Investigating higher order fits seems useless, because from the first simulation it can be concluded that the difference between 2^{nd} and 3^{rd} order fits is not significant. From earlier work done by C. Vottis [13] it can be concluded that higher order fits and more time stamps demand more calculation time. So the used fit algorithms will use 3, 4 or 5 time stamps and are of $1^{st}, 2^{nd}$ and 3^{rd} order.

Velocity

To investigate the influence of the input velocity, simulations are run for a velocity range of [10, 300] rad/s. During the simulations, the other parameters will be:

- Disturbance frequency $f = 100\ Hz$
- Sampling frequency $f_s = 10\ kHz$

Fig. 4.3 shows the results of these simulations. It can be seen that 1st order fits are in comparison with the 2^{nd} and 3^{rd} order fits performing worse and are therefore for these velocities not favourable in position estimation. This observation is comparable to the one C. Vottis made [13]. Also, from this figure one can conclude that at low speeds the algorithms produce relatively large errors. At low speeds, there has not always been a quadrature event every sample period. Therefore, there's no new polynomial fit calculated and the position is estimated on the basis of relative "old" time stamps, but with 2^{nd} and 3^{rd} fits it is still more accurate than measuring the position without time stamping. Furthermore, it seems like taking more time stamps into account deteriorates the position estimation. If too much history of the input signal is considered, the slope of the polynomial fit and the input signal sometimes don't. This causes large errors. Finally from this graph it is clear that, at velocities larger than 50 rad/s, the difference between 2^{nd} and 3^{rd} order fits in not significant.

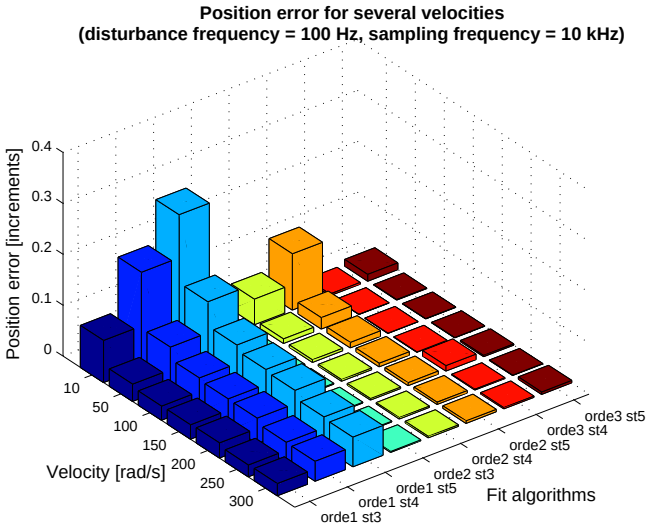


Figure 4.3: Varying the angular velocity

Disturbance frequency

The next parameter that is investigated is the disturbance frequency on the input signal. Simulations are run for a disturbance range of [25, 1000] Hz. During the simulations, the other parameters will be:

- Angular velocity $v = 250\ rad/s$

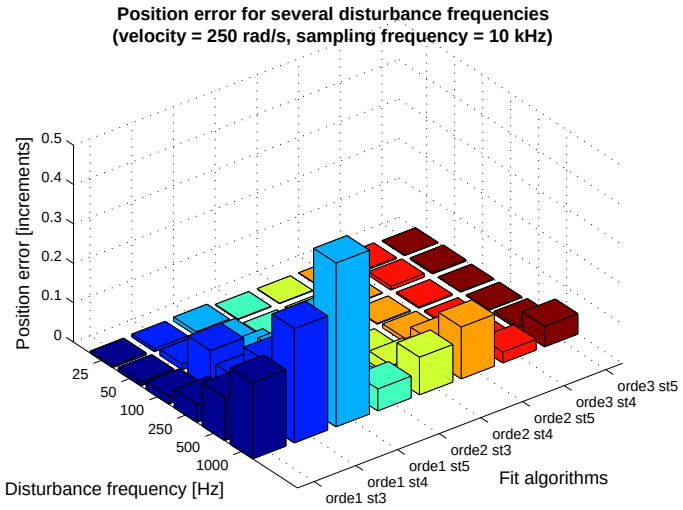


Figure 4.4: Varying the disturbance frequency

- Sampling frequency $f_s = 10\ kHz$

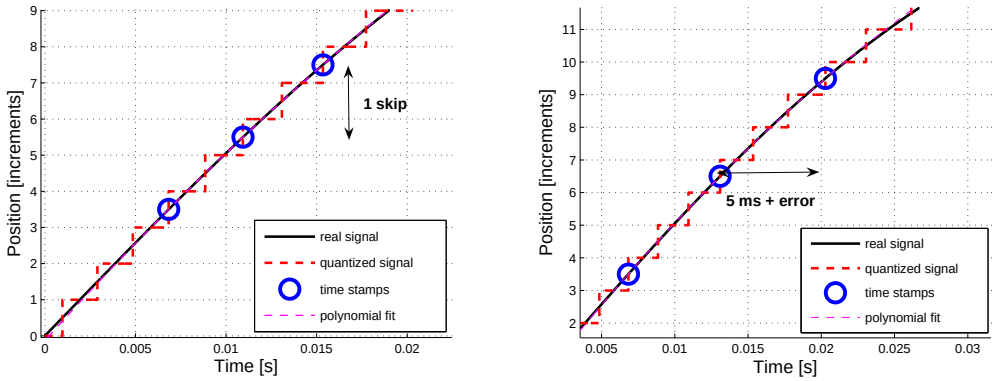
Fig. 4.4 shows the results of these simulations. First thing to notice is that 1st order fits, again, perform worse. These algorithms can only cope with disturbance frequencies up to 50 Hz, because at higher frequencies the position error doesn't differ much of the error when using the quantized signal. The 2^{nd} and 3^{rd} order fits perform better (up to 250 Hz and 500 Hz, respectively). Another conclusion from these results, as well as the results of the simulation where velocity is varied, is that when more time stamps are used to calculate the fit, the position error is larger. In [13] C. Vottis draws no explicit conclusions on this matter.

4.2.3 Skip and time delay

The skip option and the time delay option also have to be investigated. The objective is to gain insight in these options by means of simulations. These options are present on the TUeDACS/1 MicroGiant, because in some situations they may improve the accuracy of the position estimation by taking into account more history of the input signal. Fig. 4.5(a) and 4.5(b) below show the concept of both options. Notice that the time between two successive time stamps in the time delay mode is not exactly Δt, but $\Delta t + \varepsilon$, because of the time ε between the end of Δt and the time instant of the next quadrature event that occurs in time.

In the remainder of this section the following questions will be answered. In which situations do these options increase the accuracy of the position estimation? How many time stamps have to be ignored in the skip option? What should be the time delay?

In order to get answers to this kinds of questions, some various velocity profiles are investigated. The profiles differ from each other in nominal velocity, disturbance amplitude and disturbance frequency. In Appendix B results are shown for simulations with the various velocity profiles. The position error is calculated for different a number of skips, time delays and fit algorithms. From the results of Appendix B follows that if the number of skips gets

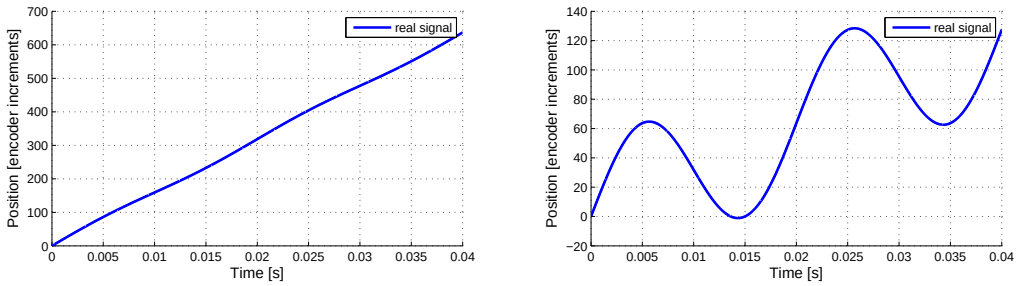


(a) Skip mode (number of skips = 1)

(b) Time delay mode (time delay = 5 ms)

Figure 4.5: Concept of time stamping modes

larger, the error between the given position and the position estimated by the polynomial fit also gets larger. The same trend applies for time delays. In fact, there are only two velocity profiles in which an improvement can be noticed. These profiles are shown in figures 4.6(a) and 4.6(b).

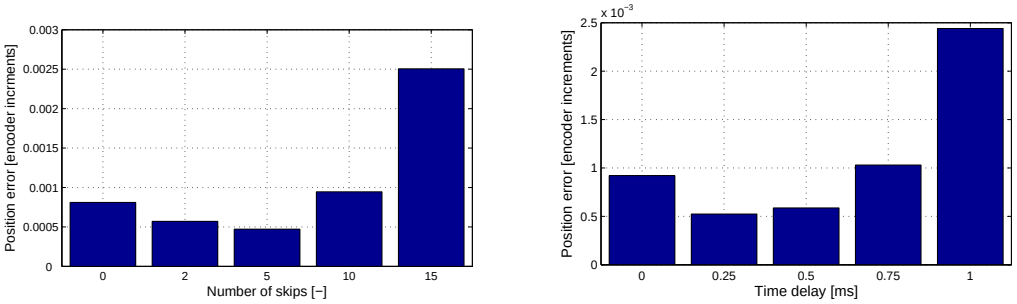


(a) $v_{nom} = 250\ rad/s, A_{dist} = 0.1, f_{dist} = 50\ Hz$ (b) $v_{nom} = 50\ rad/s, A_{dist} = 0.75, f_{dist} = 50\ Hz$

Figure 4.6: Velocity profiles

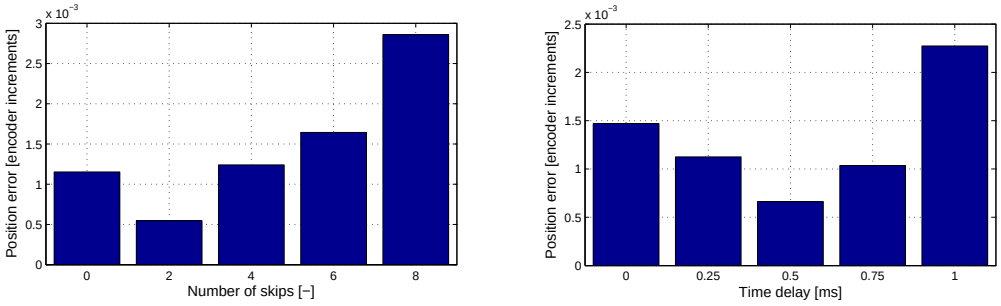
They have in common that both profiles contain a lot of quadrature events per period of the disturbance. When a 3^{rd} order polynomial fit with 5 time stamps is used to estimate the position of profile 4.6(a), the errors are like shown in figures 4.7(a), 4.7(b). In figures 4.8(a), 4.8(b) the same is done for the other velocity profile, but with a 4^{th} order polynomial fit.

As one can see from the Fig. 4.7(a), 4.7(b), 4.8(a) and 4.8(b), the rms-value of the position error can be reduced to approximately half it's value when no skip or time delay is used. The improvement by using skip or time delay is achieved when 2 - 5 skips or 0,25 - 0,5 ms time delay are applied. This corresponds to approximately 1 - 2 % of the disturbance period.



(a) number of skips (b) time delays

Figure 4.7: Error profile 4.6(a) for several skips and time delays



(a) number of skips (b) time delays

Figure 4.8: Error profile 4.6(b) for several skips and time delays

Chapter 5

Implementation

The subject of this chapter is the implementation of the time stamping concept. Experiments will be done to check the performance of the concept on a experimental setup. First, a description of the experiment will be given. Next, an overview of the experimental set-up, used for the experiments, is described in Section 5.2. Finally, some results will be presented in Section 5.3.

5.1 Description of the experiment

To evaluate the performance of the time stamping concept, the following experiment is performed. Time stamps from optical incremental encoder with a low resolution are collected. Using these time stamps, polynomial fits are calculated from which the position can be estimated. Now, the position estimations determined by the polynomial fit are compared to the quantized position from the low resolution encoder. As a reference to determine the improvement in accuracy, an encoder with a high resolution will be used. In the following paragraph the experimental set-up will be discussed.

5.2 Overview of the experimental set-up

The experimental set-up is a series of components. A mechanical part, consisting of a dc electric motor with two encoders attached to it, an amplifier, a TUeDACs/1 MicroGiant data acquisition system and a notebook are connected to each other. A block diagram of all components can be seen in Fig. 5.1.

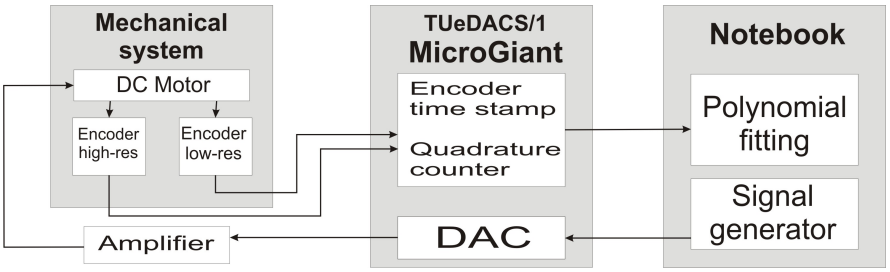
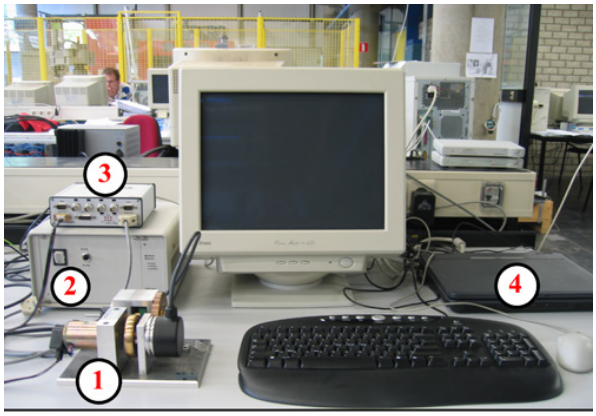


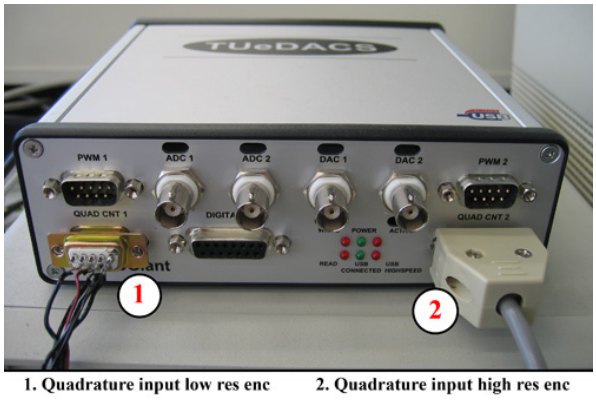
Figure 5.1: Block diagram of the experimental set-up

In Fig. 5.2(a), 5.2(b) and 5.2(c) some photos of the instruments that were used, are shown.



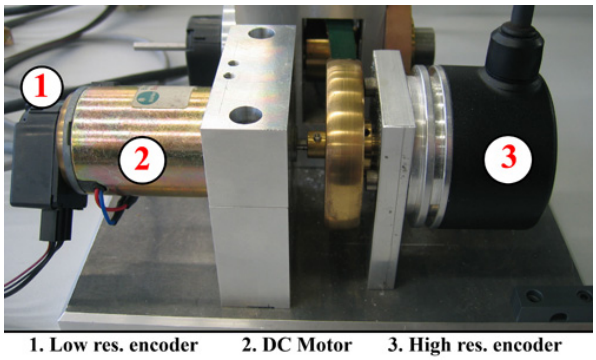
1. Mechanical system 2. Amplifier 3. MicroGiant 4. Notebook

(a) Overview of the experimental set-up



1. Quadrature input low res enc 2. Quadrature input high res enc

(b) TUeDACS/1 MicroGiant



1. Low res. encoder 2. DC Motor 3. High res. encoder

(c) Mechanical system

Figure 5.2: Experimental set-up

5.2.1 The mechanical part

The mechanical part used in this experimental set-up is taken over, except a few adaptations, from the project of C. Vottis [13]. It was built specially to meet the certain objectives that were necessary to carry out the validation experiments. The set-up consists of a electric DC motor with two encoders attached to it. On one side of the shaft, a Heidenhain ROD-426 optical incremental encoder is mounted. This encoder has 5000 increments per revolution. On the other side of the shaft, a HEDS 5500 encoder, with 100 increments per revolution, is mounted. For more information on the mechanical set-up, one can look into [13].

5.2.2 TUeDACS/1 MicroGiant

A the TUEeDACS/1 MicroGiant is used for the data acquisition purposes. It counts the number of quadrature events and as already mentioned before in Chapter 3, it has the possibility to store the time instant at which quadrature events take place. Furthermore, the device sends out the input signal the DC motor by means of a DAC output (not connected in Fig 5.2(b).

The main functions that are used in this project are a 32-bit quadrature counter and a 50-bit time stamp generator. The quadrature counter has a maximum count frequency of 10^7 $counts/s$ and time stamps are generated with a resolution of 50 ns. As the time stamps are 50 bits wide, the MicroGiant acquisition can continue without a time stamp overflow for a period of $2^{50} \cdot 50$ ns = 56,294,995.34 s, which is approximately 651.5 days.

The MicroGiant records each change of the quadrature count value as well as the time stamp at which corresponding change has occurred. When the first change occurs at t_n, the 32-bit quadrature count is stored in the Quadrature Count Register 1 and the 50-bit time stamp is stored in the Timestamp Register 1. When the next change occurs at t_{n+1}, the contents of Quadrature Count Register 1 and Timestamp Register 1 move to Quadrature Count Register 2 and Timestamp Register 2. The new quadrature count value and the corresponding time stamp are stored in Quadrature Count Register 1 and Timestamp Register 1. This storage principle is illustrated in Fig. 5.3.

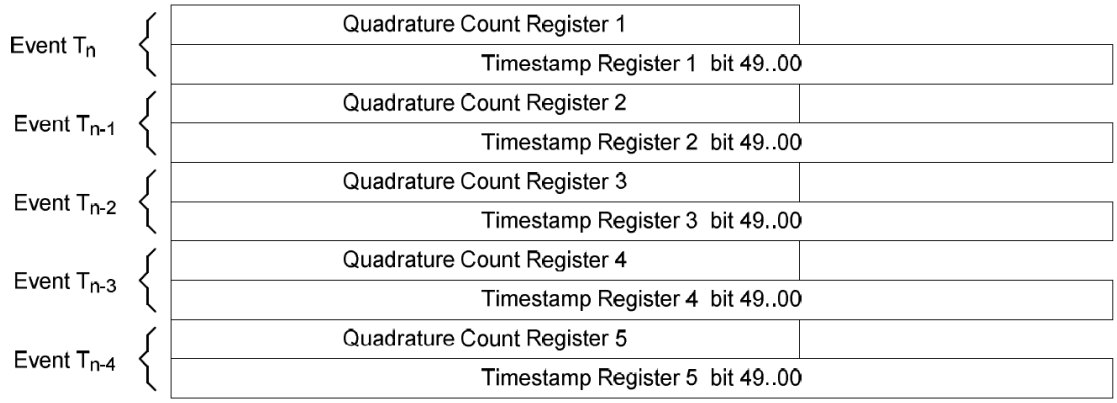
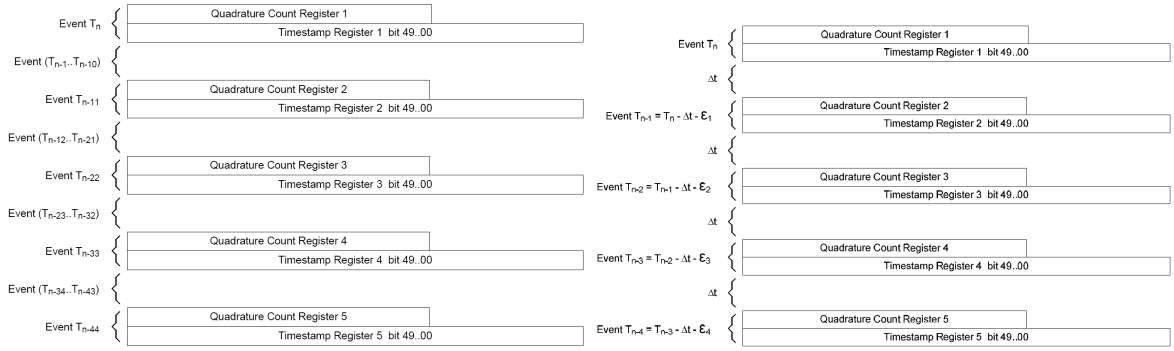


Figure 5.3: The storage principle

It is possible to use the additional skip and time delay option. When one of these option is enabled, the storage principle is changed. Fig. 5.4(a) and 5.4(b) show the adapted storage principles in skip or time delay mode.



(a) skip mode (b) time delay mode

Figure 5.4: Storage principles

With the MicroGiant it is possible to store a maximum of 5 time stamps, the skip value can be increased up to 65536 and the delay time can be 6.7 s with a resolution of 102.4 μs. For a more detailed description about the TUeDACS/1 MicroGiant see the user manual [14].

5.3 Experiments

5.3.1 Initial experiments

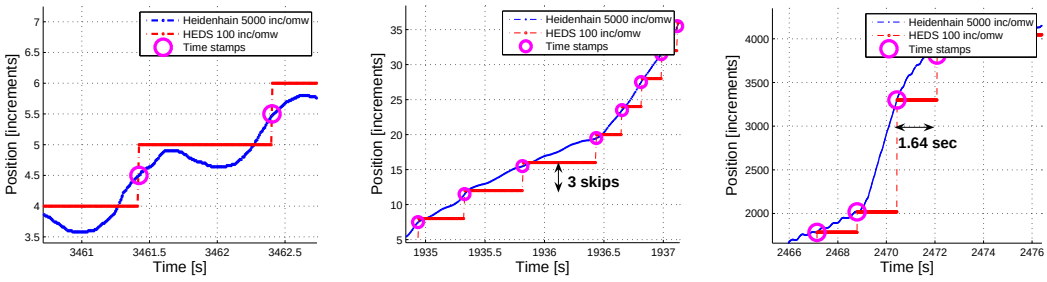
For the time being, the encoder time stamp section of the TUeDACS/1 MicroGiant is only run in simulation mode, which means that artificial encoder counts were used to develop the time stamp section. Therefore, some initial experiments were done. The experiments were carried out on a Linux operating system. Using Linux, a sample frequency of approximately 100 Hz was feasible. With the initial experiments is checked wether the different modes of the TUeDACS/1 Microgiant are working properly. There were also experiments carried out to validate the fit algorithms.

Checking different time stamping modes

Fig. 5.5(a), 5.5(b) and 5.5(c) show the time stamps that are generated by the MicroGiant in normal, skip and time delay mode, respectively. In the skip mode, a skip value of 3 was inserted. In the time delay mode, a time delay value of 65536, which should correspond to a time delay $\Delta t = 6.7$ s is used. However, from Fig. 5.5(c) can be seen that the time delay is only 1.64 s. So, the resolution isn't 102.4 μs as described in the user manual, but 25 μs approximately.

Checking different fit algorithms

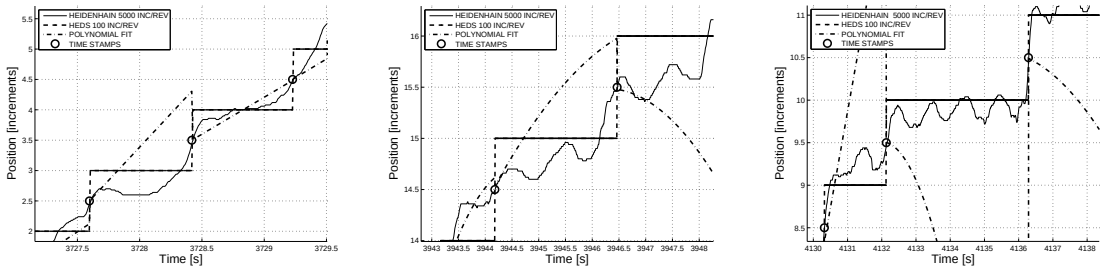
Next, the fit algorithms are checked. Figure 5.6(a), 5.6(b) and 5.6(c) show the extrapolations with $1^{st}, 2^{nd}$ and 3^{rd} order polynomial fits. As one can see the shape of the extrapolation is as expected. However, the polynomial fits show some unexpected behaviour. In Fig. 5.7(a) a



(a) Normal mode (b) Skip mode (c) Delay mode

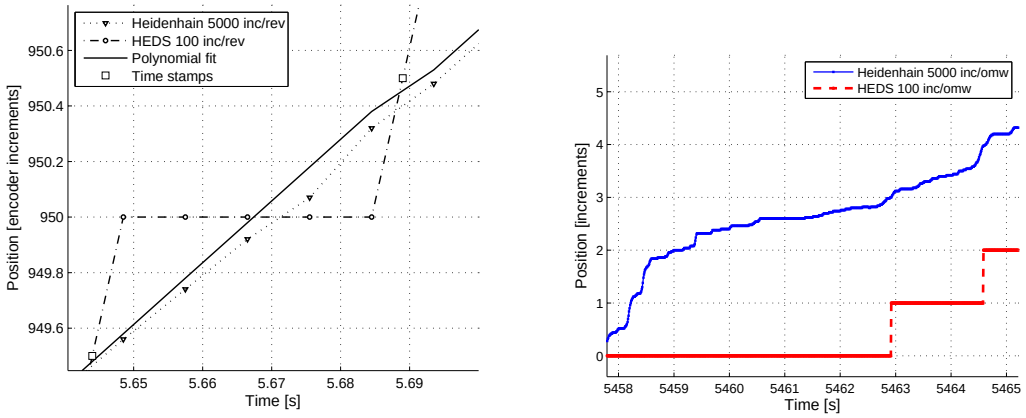
Figure 5.5: Time stamping modes

detail can be seen of a 2^{nd} order polynomial fit which is determined out of 3 time stamps, which should be an exact fit. Now the extrapolation doesn't exactly intersect the time stamps. This possible truncation error may be explained by finite accuracy of float-types in determining the polynomial coefficients. In future research this problem should be investigated.



(a) 1st order (b) 2nd order (c) 3rd order

Figure 5.6: Fit algorithms



(a) Truncation error (b) Play error

Figure 5.7: Encountered problems

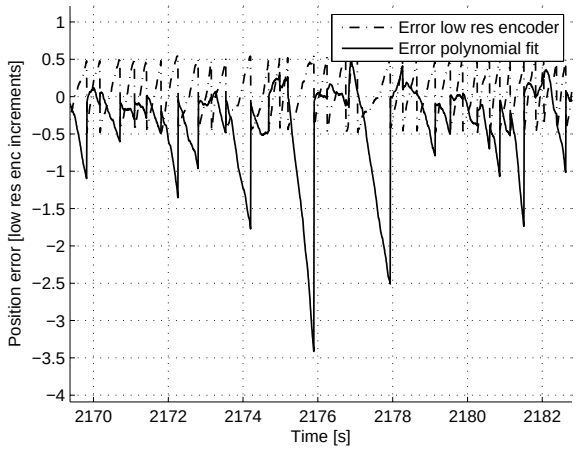
Another problem, shown in Fig. 5.7(b), is that the number of increments counted by the low resolution encoder and the ones counted by the high resolution encoder initially do not match. Reason for the initial shift could be the dynamical behaviour of the mechanical system. There may be a phase shift between the one end of the shaft, where the low res encoder is mounted, and the other end, where the high res encoder is mounted. When moving at low velocities, this phase shift should disappear. When doing so, there is still a shift in counted encoder increments. The initial shift also could be caused by play in the DC motor and then seems to be inevitable.

5.3.2 Position estimation

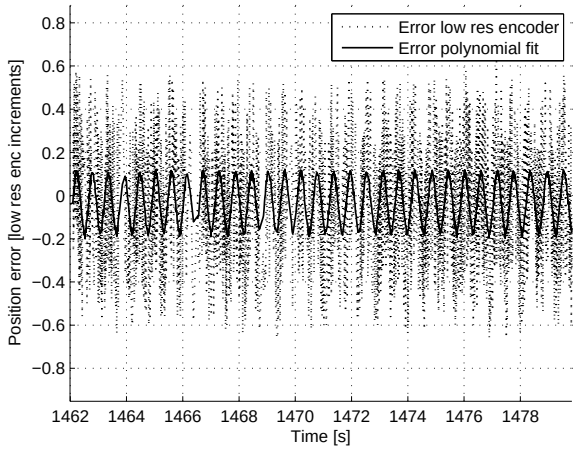
Constant velocity profiles

Now it's possible to run experiments to investigate the improvements on position estimation with time stamping. First, a constant velocity is chosen as input signal on the DC motor. For different kinds of angular velocity, the errors of the quantized signal and the position estimation based on time stamping are shown in Fig. 5.8(a), 5.8(b) and 5.8(c). The 2^{nd} order fit algorithm has been implemented, because this algorithm is the best assessment between calculation time and accuracy. During this experiment the fits were calculated using 5 time stamps.

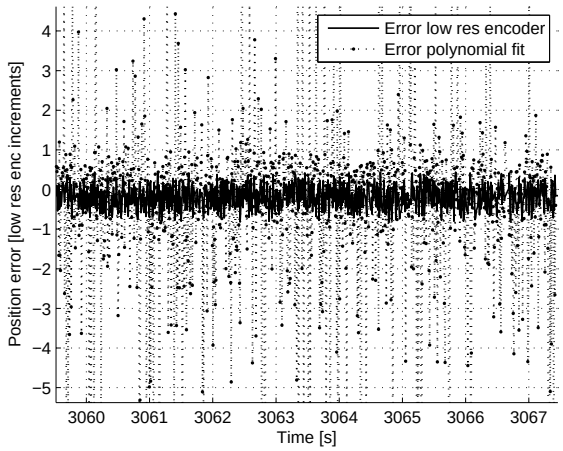
A few observations can be extracted from these figures. First, at an angular velocity of 3 inc/s, the position estimation made by the polynomial fit is not very accurate. This is caused by the lack of time stamp generation. If no new time stamp is generated, no new polynomial fit can be calculated. The position is estimated with relative "old" polynomial coefficients and therefore not accurate. If the position deviation gets larger than 0.5 low resolution encoder increments, using the quantized position is a better option. Next, at an angular velocity of 700 inc/s, the fit reduces the position error to only 30 - 40 % of the error when the quadrature signal of the low resolution encoder is used. The sinusoidal shape of the position error is probably caused by an shaft eccentricity. At first sight, the rotation frequency of the shaft equals the frequency that is present in the error. Finally, at an angular velocity of 35000 inc/s the position estimation from the polynomial fit very inaccurate. The fit totally miscalculates the position with errors up to 10 or 15 increments. Due to lack of time, no further investigation is done on this matter, but the cause of the large errors probably may be found in the time to calculate the polynomial coefficients.



(a) 3 inc/s



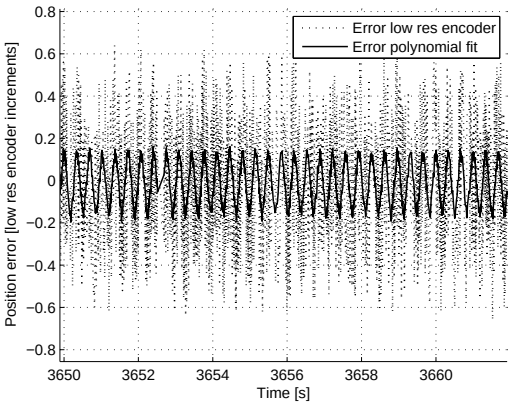
(b) 700 inc/s



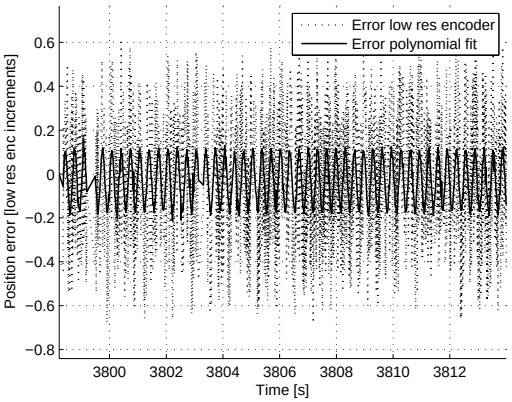
(c) 35000 inc/s

Figure 5.8: Position error for several angular velocities

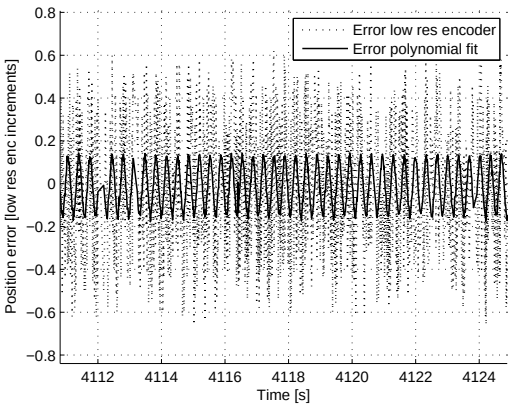
Next, an experiment is done with different fit algorithms. In Fig. 5.9(a), 5.9(b) and 5.9(c) the results are shown of $1^{st}, 2^{nd}$ and 3^{rd} order position estimations for a constant angular velocity of 700 inc/s. As one can see from these figures, there is hardly any difference between the polynomial fits.



(a) 1st order



(b) 2nd order

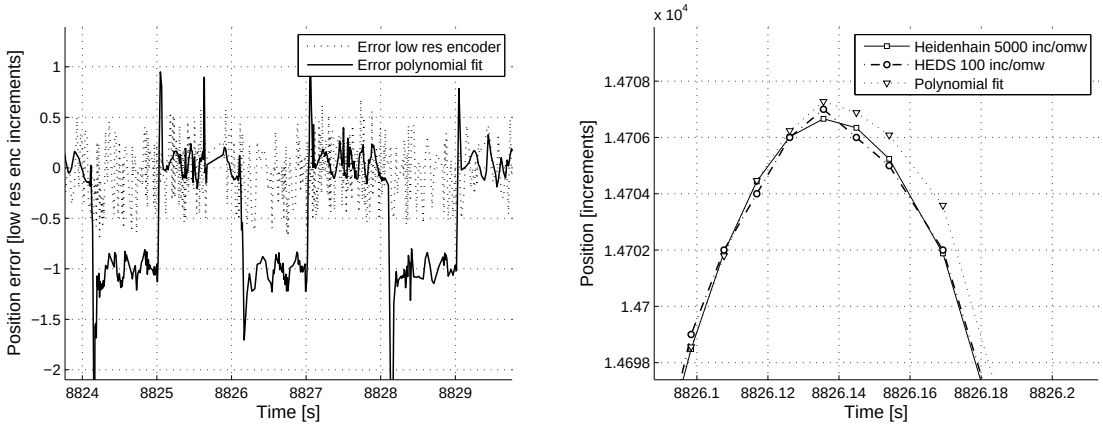


(c) 3rd order

Figure 5.9: Position estimation with 3 different fits

Sinusoidal velocity profile

As well as polynomial fitting at constant velocity profiles, polynomial fitting at sinusoidal velocity profiles must be investigated. In Fig. 5.10(a) the errors are shown of an sinusoidal velocity with a frequency of 0.5 Hz. However, a strange phenomenon is present. When the shaft starts rotating in the opposite direction the polynomial fit miscalculates the position in such way as is shown in Fig. 5.10(b). This phenomenon could not be explained at the moment and thus, sinusoidal velocity profiles are left out of consideration.



(a) Position error at sinusoidal velocity

(b) Miscalculation

Figure 5.10: Sinusoidal velocity profile

5.3.3 Velocity estimation

Not only position estimation can be improved by time stamping, but also velocity estimation is a significant advantage of time stamping. To estimate the velocity one should determine the derivative of the polynomial that is used for position estimation. So, the velocity, the derivative of Equation 4.6, would be

$$v = mc_m t^{m-1} + (m-1)c_{m-1}t^{m-2} + \cdots + c_1. \tag{5.1}$$

Fig. 5.11 shows the velocity estimation of the polynomial fit, low resolution encoder and high resolution encoder. Last mentioned serves as reference velocity. The backward difference method is used to determine the velocity of the quantized signals.

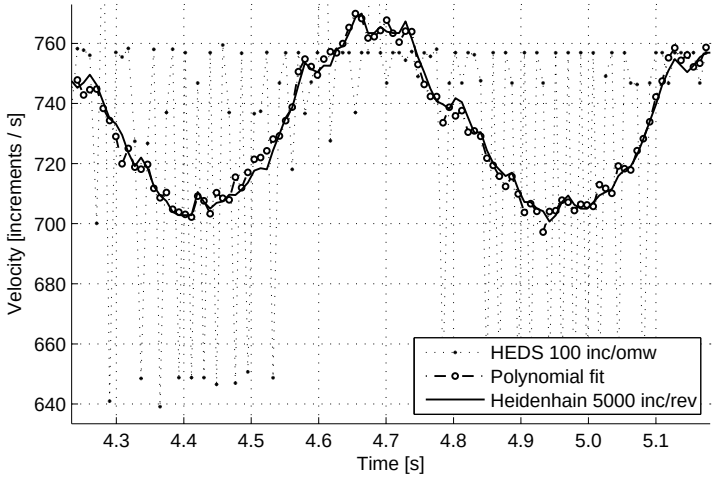


Figure 5.11: Velocity estimation

Chapter 6

Conclusion

6.1 Conclusion

An extensive literature search has been carried out for existing methods to improve position and velocity estimations using incremental optical encoders. Over the past few years, many methods passed by with their advantages and disadvantages (see Chapter 2). Position estimation based on time stamping was experienced as a whole new approach because other methods don't use the combination of time instants and counter values.

In the next phase of the project, simulations were done. The purpose of the simulations was to test what the influence was of several parameters, like velocity, disturbances and fit algorithms. Also, skip and time delay options were investigated. From Chapter 4 can be concluded that fitting a low order ($1^{st}, 2^{nd}$ or 3^{rd}) polynomial through time stamps, which are generated at quadrature events, improves the accuracy of the position estimation. Because during the simulations the controller sampling frequency was set at 10 kHz the resulting errors are a bit optimistic. But from the simulations that were done the position error can be reduced to hundredths of an encoder step. It seems that the 3^{rd} order polynomial performed best, but a 2^{nd} order polynomial was not significantly worse. Due to a shorter calculation time, the latter one may be more favourable. Furthermore, it can be concluded from the simulations that skip and time delay options are only advisable to use in situations where a lot of quadrature events take place ($100 +$) within a period of disturbance and the disturbance frequency should be low.

The time stamping concept was implemented on an experimental set-up. C code was written to

- communicate with the TUeDACS/1 MicroGiant, which was responsible for generating the time stamps, and to
- calculate the polynomial fits, from which the position and velocity estimation was determined.

Different fit algorithms were applied on the time stamps to estimate the position and velocity for constant velocity profiles. Dependent on the angular velocity of the DC motor, the results were satisfying. Mainly, at speeds up and until approximately 1000 inc/s the position error could be reduced to 30 - 40 % of the quantization error. The velocity estimation is even more accurate.

Although some experiments were done with skip and time delay modes, only the functionality on the MicroGiant are discussed. Due lack of time to investigate these modes no sensible statements could be mentioned.

6.2 Recommendations

Before implementing the time stamp concept in precision motion systems, a few recommendations for further investigation are to be mentioned.

First of all, the cause of the initial shift should be declared. A possible explanation is mentioned, but this doesn't necessary have to be the real cause.

Next, the cause of the non-intersecting polynomial fit through time stamps at conditions where an exact fit should have been calculated has to be found. Just like in case of the initial shift a possible explanation is discussed. The finite accuracy of float-types could be the cause. Mainly, because the polynomial coefficients in Matlab were calculated in the right way.

Furthermore, the fit algorithms need to be optimized for relative high velocities and for sinusoidal movement of the DC motor. These problems were mentioned in Chapter 5.

To achieve solutions to these problems, a experimental set-up without a shaft eccentricity may be helpful.

Appendix A

M-files used for simulations

A.1 Normal mode

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%             MicroGiant Time-stamping            %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long;
clear all;
close all;

%DECLARATION OF VARIABLES
dt  = 50e-9;                                       % time stamp resolution [1/s]
v   = 10;                                          % angular velocity [rad/s]
A   = 0.05;                                        % amplitude
f   = 500;                                         % frequency [Hz]
te  = 0.002;                                       % ending time
t   = [0:dt:te];                                   % time vector

inc = 400;                                         % encoder resolution
dx  = 1;                                           % encoder step
rsp = 1e-4;                                        % sample period

n   = (te/dt)+1;                                   % number of time samples
o   = 2;                                           % order of polynomial fit
nts = 5;                                           % numbers of timestamp

control_time = 0:rsp:te;                           % control time instants
err_f_vector = [];                                 % xfit - xreal
err_q_vector = [];                                 % xquan - xreal

x   = zeros(n,1);                                  % simulated real signal
x1  = zeros(n,1);                                  % extra real signal for plotting
xq  = zeros(n,1);                                  % simulated quantized signal
xq1 = zeros(n,1);                                  % extra quantized signal for plotting
xf  = zeros(1,n);                                  % estimated signal

TS = [];                                           % time stamp array
XS = [];                                           % position array
controlinstant_vector = [];                        % sample times array
RMS_value = [];                                    % rms error estimation
RMS_value2 = [];                                   % rms error quantization

%PROGRAM EXECUTING
%constructing real signal
for i=1:n
    x1(i) = (v*t(i)+A*sin(2*pi*f*t(i)))*(inc/(2*pi));
    xq1(i)= round(x1(i)/dx)*dx;
```

```
end

%quantization + polynomial fitting
i=0;
for i=1:n
    x(i) = (v*t(i)+A*sin(2*pi*f*t(i)))*(inc/(2*pi));
    xq(i)= round(x(i)/dx)*dx;
    if i ~= 1
        if length(TS) >= nts
            TS = TS(:,2:nts);
            XS = XS(:,2:nts);
        end
        if round(x(i)/dx) ~= round((x(i-1))/dx)
            ts = t(i);
            TS = [TS ts];
            if round(x(i)/dx) > round((x(i-1))/dx)
            xs = (xq(i)-0.5*dx);
            XS = [XS xs];
            end
            if round(x(i)/dx) < round((x(i-1))/dx)
            xs = (xq(i)+0.5*dx);
            XS = [XS xs];
            end
        end
        if length(TS) == nts
            cf  = coefflu(TS,XS,o,nts)
            xf  = polyval(cf,t);
            controlinstant_ = (floor((i*dt)/rsp) + 1)*rsp;
            controlinstant__vector = [controlinstant__vector controlinstant_];
            controlinstant__i = int32(((floor((i*dt)/rsp) + 1)*rsp)/dt + 1);
            err_f = ((x1(controlinstant__i) - polyval(cf,controlinstant_)));
            err_f_vector = [err_f_vector err_f];
            err_q = ((x1(controlinstant__i)  - xq1(controlinstant__i)));
            err_q_vector = [err_q_vector err_q];
            plotapprox(t,x1,xq1,TS,XS,xf,control_time);
        end

        if i==n
                TS = [];
                XS = [];
                timer = 0.0;
        end
    end
end

RMS_value = [RMS_value rms(err_f_vector)];
RMS_value2 = [RMS_value2 rms(err_q_vector)];
ploterror(controlinstant__vector, err_f_vector, err_q_vector, dx);
```

A.2 Skip mode

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    MicroGiant Time-stamping with Skip option    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long;
clear all;
close all;

%DECLARATION OF VARIABLES
dt  = 50e-9;                                        % time stamp resolution [1/s]
v   = 10;                                           % angular velocity [rad/s]
A   = 0.05;                                         % amplitude
f   = 500;                                          % frequency [Hz]
te  = 0.002;                                        % ending time
```

```
t   = [0:dt:te];                                % time vector

inc = 400;                                      % encoder resolution
dx  = 1;                                        % encoder step
rsp = 1e-4;                                     % sample period

n   = (te/dt)+1;                                % number of time samples
o   = 2;                                        % order of polynomial fit
nts = 5;                                        % numbers of timestamp

counter = 0;                                    % counter to count quadrature events
control_time = 0:rsp:te;                        % control time instants
err_f_vector = [];                              % xfit - xreal
err_q_vector = [];                              % xquan - xreal

x   = zeros(n,1);                               % simulated real signal
x1  = zeros(n,1);                               % extra real signal for plotting
xq  = zeros(n,1);                               % simulated quantized signal
xq1 = zeros(n,1);                               % extra quantized signal for plotting
xf  = zeros(1,n);                               % estimated signal

TS = [];                                        % time stamp array
XS = [];                                        % position array
controlinstant_vector = [];                     % sample times array
RMS_value = [];                                 % rms error estimation
RMS_value2 = [];                                % rms error quantization
controlinstant_old = 1e-4;                      % start value of first control instant



%PROGRAMMA UITVOERING
%constructing real signal
for i=1:n
    x1(i) = (v*t(i)+A*sin(2*pi*f*t(i)))*(inc/(2*pi));
    xq1(i)= round(x1(i)/dx)*dx;
end


for skip = [1]

    controlinstant_vector =[1e-4];
    err_f_vector = [1e-6];
    err_q_vector = [];
    i=0;
    for i=1:n
        x(i) = (v*t(i)+A*sin(2*pi*f*t(i)))*(inc/(2*pi));
        xq(i)= round(x(i)/dx)*dx;
        if i ~= 1
            if length(TS) >= nts
                TS = TS(:,2:nts);
                XS = XS(:,2:nts);
            end

            if (round(x(i)/dx) ~= round((x(i-1))/dx)) & (teller == skip)
                counter = 0;
                ts = t(i);
                TS = [TS ts];
                if round(x(i)/dx) > round((x(i-1))/dx)
                xs = xq(i)-0.5*dx;
                XS = [XS xs];
                end
                if round(x(i)/dx) < round((x(i-1))/dx)
                xs = xq(i)+0.5*dx;
                XS = [XS xs];
                end
            elseif (round(x(i)/dx) ~= round((x(i-1))/dx)) & (teller ~= skip)
```

```
            counter = counter + 1;
        else
            counter = counter;
        end

        if length(TS) == nts
            cf  = coefflu(TS,XS,o,nts);
            xf  = polyval(cf,t);
            xf2 = polyval(cf,control_time);
            controlinstant = (floor((i*dt)/rsp) + 1)*rsp;

            if controlinstant == controlinstant_old
                controlinstant_i = int32(((floor((i*dt)/rsp) + 1)*rsp)/dt + 1);
                err_f = abs((x1(controlinstant_i) - polyval(cf,controlinstant)));
                k = length(err_f_vector);
                err_f_vector(k) = err_f;
                controlinstant_old = controlinstant;
            else
                controlinstant_vector = [controlinstant_vector controlinstant];
                controlinstant_i = int32(((floor((i*dt)/rsp) + 1)*rsp)/dt + 1);
                err_f = abs((x1(controlinstant_i) - polyval(cf,controlinstant)));
                err_f_vector = [err_f_vector err_f];
                controlinstant_old = controlinstant;
            end
            err_q = abs((x1(controlinstant_i)  - xq1(controlinstant_i)));
            err_q_vector = [err_q_vector err_q];
            plotapprox(t,x1,xq1,TS,XS,xf,control_time);
        end

        if i==n
            TS = [];
            XS = [];
            counter = 0;
        end
    end
end

ploterror(controlinstant_vector, err_f_vector,t,xq1);
RMS_value = [RMS_value rms(err_f_vector)];
end
```

A.3 Time delay mode

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   MicroGiant Time-stamping with Delay option    %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long;
clear all;
close all;

%DECLARATION OF VARIABLES
dt  = 50e-9;                                    % time stamp resolution [1/s]
v   = 10;                                       % angular velocity [rad/s]
A   = 0.05;                                     % amplitude
f   = 500;                                      % frequency [Hz]
te  = 0.002;                                    % ending time
t   = [0:dt:te];                                % time vector

inc = 400;                                      % encoder resolution
dx  = 1;                                        % encoder step
rsp = 1e-4;                                     % sample period

n   = (te/dt)+1;                                % number of time samples
o   = 2;                                        % order of polynomial fit
```

```
nts = 5;                                          % numbers of timestamp

timer = 0;                                        % timer to keep up elapsed time
control_time = 0:rsp:te;                          % control time instants
err_f_vector = [];                                % xfit - xreal
err_q_vector = [];                                % xquan - xreal

x   = zeros(n,1);                                 % simulated real signal
x1  = zeros(n,1);                                 % extra real signal for plotting
xq  = zeros(n,1);                                 % simulated quantized signal
xq1 = zeros(n,1);                                 % extra quantized signal for plotting
xf  = zeros(1,n);                                 % estimated signal

TS = [];                                          % time stamp array
XS = [];                                          % position array
controlinstant_vector = [];                       % sample times array
RMS_value = [];                                   % rms error estimation
RMS_value2 = [];                                  % rms error quantization
controlinstant_old = 1e-4;                        % start value of first control instant

%PROGRAMMA UITVOERING
%constructing real signal
for i=1:n
    x1(i) = (v*t(i)+A*sin(2*pi*f*t(i)))*(inc/(2*pi));
    xq1(i)= round(x1(i)/dx)*dx;
end

for delay = [0.005]

    controlinstant_vector =[1e-4];
    err_f_vector = [1e-6];
    err_q_vector = [];
    i=0;
    for i=1:n
        x(i) = (v*t(i)+A*sin(2*pi*f*t(i)))*(inc/(2*pi));
        xq(i)= round(x(i)/dx)*dx;
        if i ~= 1
            if length(TS) >= nts
                TS = TS(:,2:nts);
                XS = XS(:,2:nts);
            end

            if (round(x(i)/dx) ~= round((x(i-1))/dx)) & (timer >= delay)
                timer = 0;
                ts = t(i);
                TS = [TS ts];
                if round(x(i)/dx) > round((x(i-1))/dx)
                xs = xq(i)-0.5*dx;
                XS = [XS xs];
                end
                if round(x(i)/dx) < round((x(i-1))/dx)
                xs = xq(i)+0.5*dx;
                XS = [XS xs];
                end
            elseif (round(x(i)/dx) ~= round((x(i-1))/dx)) & (timer < delay)
                timer = timer + dt;
            else
                timer = timer + dt;
            end

            if length(TS) == nts
                cf  = coefflu(TS,XS,o,nts);
                xf  = polyval(cf,t);
                xf2 = polyval(cf,control_time);
                controlinstant = (floor((i*dt)/rsp) + 1)*rsp;
```

```
                if controlinstant == controlinstant_old
                    controlinstant_i = int32(((floor((i*dt)/rsp) + 1)*rsp)/dt + 1);
                    err_f = abs((x1(controlinstant_i) - polyval(cf,controlinstant)));
                    k = length(err_f_vector);
                    err_f_vector(k) = err_f;
                    controlinstant_old = controlinstant;
                else
                    controlinstant_vector = [controlinstant_vector controlinstant];
                    controlinstant_i = int32(((floor((i*dt)/rsp) + 1)*rsp)/dt + 1);
                    err_f = abs((x1(controlinstant_i) - polyval(cf,controlinstant)));
                    err_f_vector = [err_f_vector err_f];
                    controlinstant_old = controlinstant;
                end

                err_q = abs((x1(controlinstant_i)  - xq1(controlinstant_i)));
                err_q_vector = [err_q_vector err_q];
                plotapprox(t,x1,xq1,TS,XS,xf,control_time);
            end

            if i==n
                TS = [];
                XS = [];
                timer = 0.0;
            end
        end
    end

ploterror(controlinstant_vector, err_f_vector,t,xq1);
RMS_value = [RMS_value rms(err_f_vector)];
end
```

A.4 Determining polynomial coefficients

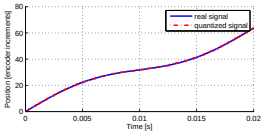
```
%POLYNOMIAL FITTING
%a = TS         time vector with time instants
%b = XS         position vector
%c = o          order of polynoimal
%e = nts        number of timestamp

function [coefflu] = fit(a, b, c, e)

    AtA = zeros(c+1,c+1);
    coef = zeros(c+1,1);
    d = zeros(c+1,1);
    z = zeros(c+1,1);
    m = zeros(c+1,c+1);

    %construct A'*A
    i = 0;
    j = 0;
    n = 0;
    for i=1:(c+1)
        for j=1:(c+1)
            for n=1:e
                AtA(i,j) = AtA(i,j) + a(n)^((c+c)-(i-1)-(j-1));
            end
        end
    end
    ata = AtA;

    %construct A'*b
    i = 0;
    j = 0;
    n = 0;
    for j=1:(c+1)
```

```
    for n=1:e
        d(j) = d(j) + (a(n)^(c-(j-1)) * b(n));
    end
end
atb = d;

%facorization
i = 0;
j = 0;
k = 0;
for k = 1:c
    for i = (k+1):(c+1)
        m(i,k) = AtA(i,k)/AtA(k,k);
    end
    for j =(k+1):(c+1)
        for i = (k+1):(c+1)
            AtA(i,j) = AtA(i,j) - m(i,k)*AtA(k,j);
        end
    end
end

%forward substitution
i = 0;
for i = 1:(c+1)
    switch c
        case 0
            z(i) = d(i);
        case 1
            z(i) = d(i) - m(i,1)*z(1);
        case 2
            z(i) = d(i) - m(i,1)*z(1) - m(i,2)*z(2);
        case 3
            z(i) = d(i) - m(i,1)*z(1) - m(i,2)*z(2) - m(i,3)*z(3);
        case 4
            z(i) = d(i) - m(i,1)*z(1) - m(i,2)*z(2) - m(i,3)*z(3) - m(i,4)*z(4);
    end
end

%backward substition
i=0;
for i = (c+1):-1:1
    switch c
        case 0
            coef(i) = z(i);
        case 1
            coef(i) = (z(i) - AtA(i,2)*coef(2))/AtA(i,i);
        case 2
            coef(i) = (z(i) - AtA(i,3)*coef(3) - AtA(i,2)*coef(2))/AtA(i,i);
        case 3
            coef(i) = (z(i) - AtA(i,4)*coef(4) - AtA(i,3)*coef(3) - AtA(i,2)*coef(2))/AtA(i,i);
        case 4
            coef(i) = (z(i) - AtA(i,5)*coef(5) - AtA(i,4)*coef(4) - AtA(i,3)*coef(3) -
            AtA(i,2)*coef(2))/AtA(i,i);
    end
end
coefflu = coef;
```

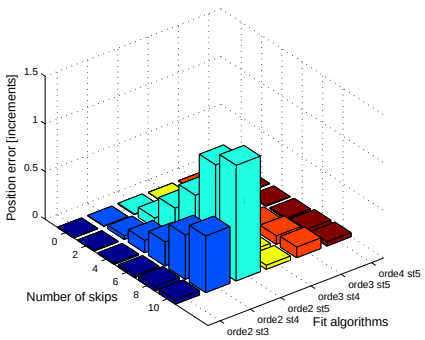
Appendix B

Results of skip and time delay simulations

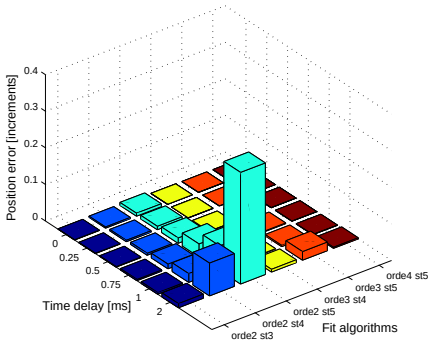
B.1 $v_{nom} = 50\ rad/s,\ A_{dist} = 0.1,\ f_{dist} = 50\ Hz$



(a) Velocity profile

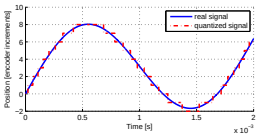


(b) Skip

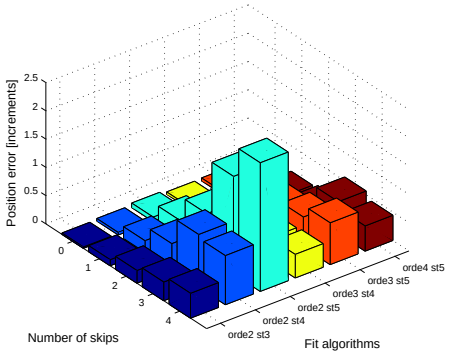


(c) Delay

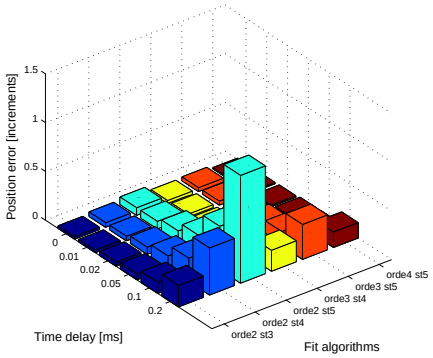
B.2 $v_{nom} = 50\ rad/s,\ A_{dist} = 0.1,\ f_{dist} = 500\ Hz$



(d) Velocity profile

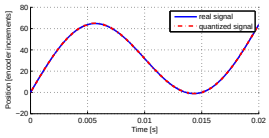


(e) Skip

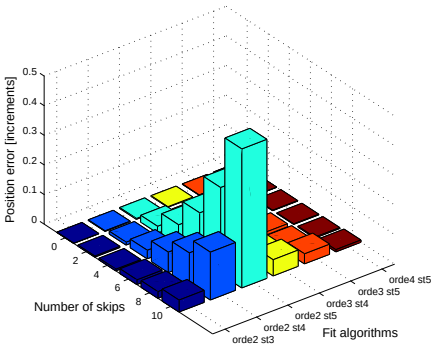


(f) Delay

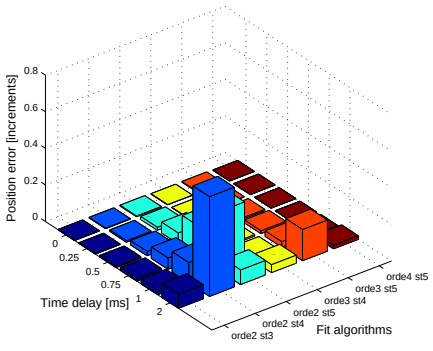
B.3 $v_{nom} = 50\ rad/s,\ A_{dist} = 0.75,\ f_{dist} = 50\ Hz$



(g) Velocity profile

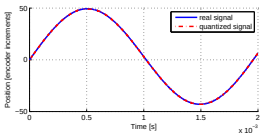


(h) Skip

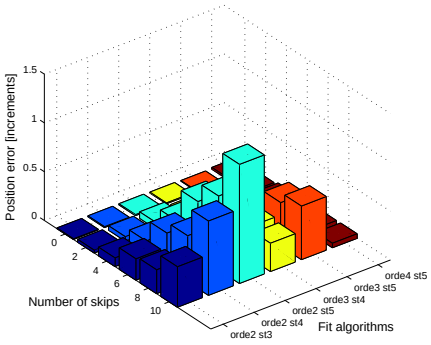


(i) Delay

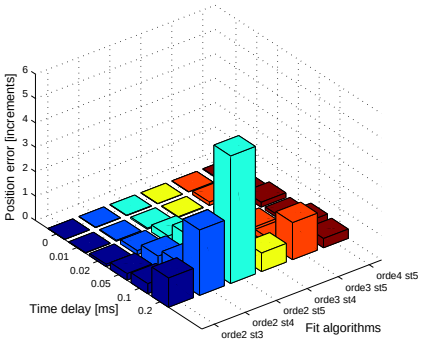
B.4 $v_{nom} = 50\ rad/s,\ A_{dist} = 0.75,\ f_{dist} = 500\ Hz$



(j) Velocity profile

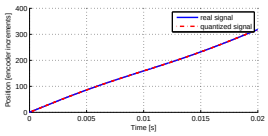


(k) Skip

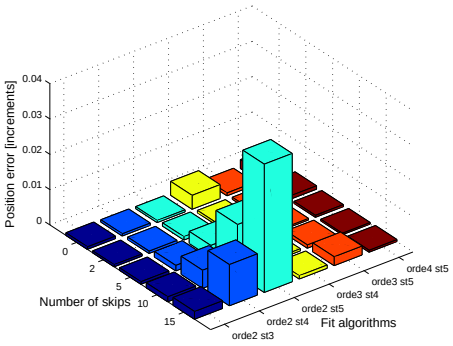


(l) Delay

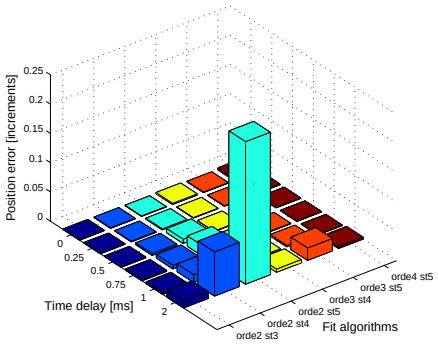
B.5 $v_{nom} = 250\ rad/s,\ A_{dist} = 0.1,\ f_{dist} = 50\ Hz$



(m) Velocity profile

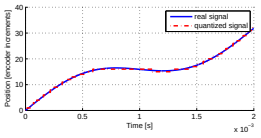


(n) Skip

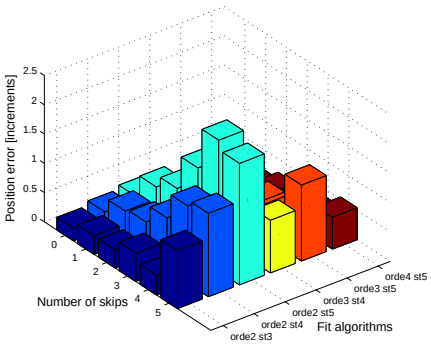


(o) Delay

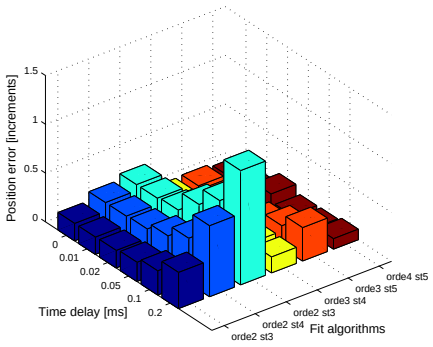
B.6 $v_{nom} = 250\ rad/s,\ A_{dist} = 0.1,\ f_{dist} = 500\ Hz$



(p) Velocity profile

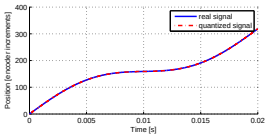


(q) Skip

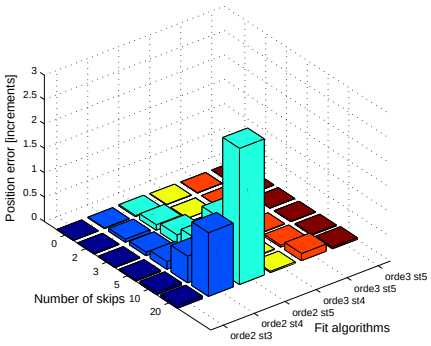


(r) Delay

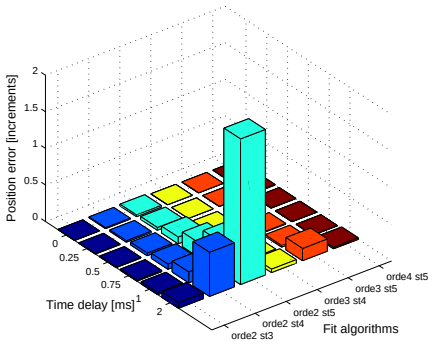
B.7 $v_{nom} = 250\ rad/s,\ A_{dist} = 0.75,\ f_{dist} = 50\ Hz$



(s) Velocity profile

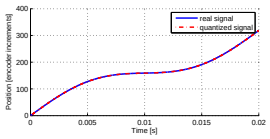


(t) Skip

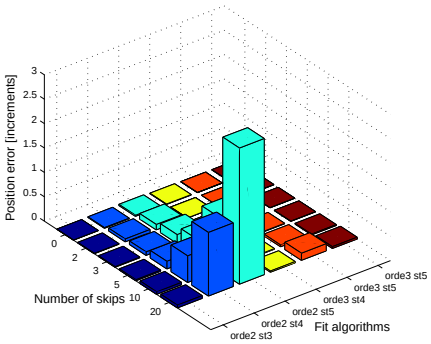


(u) Delay

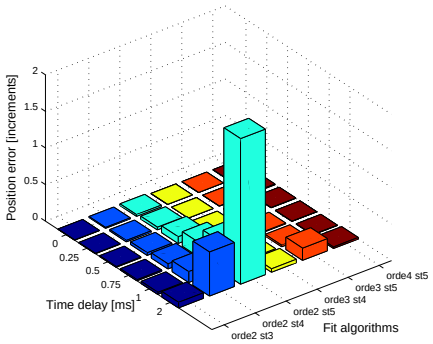
B.8 $v_{nom} = 250\ rad/s,\ A_{dist} = 0.75,\ f_{dist} = 50\ Hz$



(v) Velocity profile



(w) Skip



(x) Delay

Appendix C

C codes used for experiments

C.1 Main C code

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


#include "UTC200402tdstatus.h"
#include "utc200402tdtypes.h"
#include "io_utc200402.h"

////////////////////////////////////////////////////////////////////////
/////////////// COLLECT TIME STAMPS FROM MG//////////////////////////////
////////////////////////////////////////////////////////////////////////
main ( ){

    /* DECLARAIONS REGISTERS AND VARIABLES */
    /* REGISTERS */
    UWORD ts_status_word_register_address_ch_1 = 0x0B0;
    UWORD ts_status_word_register_address_ch_2 = 0x0D0;

    UWORD qd_trigger_mode_select_register_ch_1 = 0x051;
    UWORD qd_trigger_mode_select_register_ch_2 = 0x052;

    UWORD ts_qd_count_reg_1_ch_1 = 0x0B2;
    UWORD ts_qd_count_reg_2_ch_1 = 0x0B8;
    UWORD ts_qd_count_reg_3_ch_1 = 0x0BE;
    UWORD ts_qd_count_reg_4_ch_1 = 0x0C4;
    UWORD ts_qd_count_reg_5_ch_1 = 0x0CA;

    UWORD ts_qd_count_reg_1_ch_2 = 0x0D2;
    UWORD ts_qd_count_reg_2_ch_2 = 0x0D8;
    UWORD ts_qd_count_reg_3_ch_2 = 0x0DE;
    UWORD ts_qd_count_reg_4_ch_2 = 0x0E4;
    UWORD ts_qd_count_reg_5_ch_2 = 0x0EA;

    UWORD ts_timestamp_1_bit63_32_ch1 = 0x0B4;
    UWORD ts_timestamp_1_bit31_00_ch1 = 0x0B6;
    UWORD ts_timestamp_2_bit63_32_ch1 = 0x0BA;
    UWORD ts_timestamp_2_bit31_00_ch1 = 0x0BC;
    UWORD ts_timestamp_3_bit63_32_ch1 = 0x0C0;
    UWORD ts_timestamp_3_bit31_00_ch1 = 0x0C2;
    UWORD ts_timestamp_4_bit63_32_ch1 = 0x0C6;
    UWORD ts_timestamp_4_bit31_00_ch1 = 0x0C8;
    UWORD ts_timestamp_5_bit63_32_ch1 = 0x0CC;
    UWORD ts_timestamp_5_bit31_00_ch1 = 0x0CE;
```

```
UWORD ts_skip_delay_count_reg_ch_1 = 0x0B1;

UWORD system_time_bit63_32_reg = 0x0A4;
UWORD system_time_bit31_00_reg = 0x0A6;

UWORD ts_status1;
UWORD ts_status2;

UWORD qd_status1;
UWORD qd_status2;

UWORD ts_skip_delay1;

ULONG ts_counts1_ch1;
ULONG ts_counts2_ch1;
ULONG ts_counts3_ch1;
ULONG ts_counts4_ch1;
ULONG ts_counts5_ch1;

ULONG ts_counts1_ch2;
ULONG ts_counts2_ch2;
ULONG ts_counts3_ch2;
ULONG ts_counts4_ch2;
ULONG ts_counts5_ch2;

ULONG ts_timestamp1_bit63_32_ch1 = 0;
ULONG ts_timestamp1_bit31_00_ch1 = 0;
ULONG ts_timestamp2_bit63_32_ch1 = 0;
ULONG ts_timestamp2_bit31_00_ch1 = 0;
ULONG ts_timestamp3_bit63_32_ch1 = 0;
ULONG ts_timestamp3_bit31_00_ch1 = 0;
ULONG ts_timestamp4_bit63_32_ch1 = 0;
ULONG ts_timestamp4_bit31_00_ch1 = 0;
ULONG ts_timestamp5_bit63_32_ch1 = 0;
ULONG ts_timestamp5_bit31_00_ch1 = 0;

ULONG system_time_bit63_32;
ULONG system_time_bit31_00;


/*VARIABLES */
double timestamp1_ch1 = 0;
double timestamp2_ch1 = 0;
double timestamp3_ch1 = 0;
double timestamp4_ch1 = 0;
double timestamp5_ch1 = 0;

double system_time;
double t = 0.0;
double x = 0.0;
double v = 0.0;
double pi = 3.141592654;
double ts[5] = {0,0,0,0,0};
double xs[5] = {0,0,0,0,0};
double cf[4] = {0,0,0,0};
double *ptr_cf = cf;

int b = 0;
int n;
int i;
int nts = 5;
int o = 3;

FILE *pf;
```

```
/* PROGRAM TO COLLECT TIME STAMPS FROM TUEDACS MICROGIANT */

tdOpen( );
pf = fopen ("DATA.DAT", "w");


/*CONFIGURE QUADRATURE TRIGGER MODE REGISTER CH 1 */
tdPutWord(qd_trigger_mode_select_register_ch_1, 0x0002);

/*CONFIGURE QUADRATURE TRIGGER MODE REGISTER CH 2 */
tdPutWord(qd_trigger_mode_select_register_ch_2, 0x0002);

/* CONFIGURE TIME STAMP STATUS WORD REGISTER CH 1
   SELECT TRIGGER MODE AND PARTLY STORAGE MODE */
tdPutWord(ts_status_word_register_address_ch_1, 0x0004);

/* CONFIGURE TIME STAMP STATUS WORD REGISTER CH 1
   SELECT TRIGGER MODE AND PARTLY STORAGE MODE */
tdPutWord(ts_status_word_register_address_ch_2, 0x0004);

/* CONFIGURE SKIP AND DELAY COUNT REGISTER */
tdPutWord(ts_skip_delay_count_reg_ch_1, 0x0001);

/* ENABLE TIMESTAMPING */
tdSetBitsWord(ts_status_word_register_address_ch_1, 0x0001);


/* COLLECT TIME STAMPS */
n = 0;
while (n < 1e6){

    /* FREEZE TIME */
    tdPutWord(0xa2,0x0001);

    /* GET SYSTEM TIME */
    tdGetLong(system_time_bit63_32_reg, &system_time_bit63_32);
    tdGetLong(system_time_bit31_00_reg, &system_time_bit31_00);
    t = system_time_bit63_32 * pow(2,32)*(50e-9) +
        system_time_bit31_00 * (50e-9);

    /* READ QUADRATURE COUNT REGISTERS CH 1 */
    tdGetLong(ts_qd_count_reg_1_ch_1, &ts_counts1_ch1);
    tdGetLong(ts_qd_count_reg_2_ch_1, &ts_counts2_ch1);
    tdGetLong(ts_qd_count_reg_3_ch_1, &ts_counts3_ch1);
    tdGetLong(ts_qd_count_reg_4_ch_1, &ts_counts4_ch1);
    tdGetLong(ts_qd_count_reg_5_ch_1, &ts_counts5_ch1);

    /* READ QUADRATURE COUNT REGISTER CH 2 */
    tdGetLong(ts_qd_count_reg_1_ch_2, &ts_counts1_ch2);

    /* READ TIME STAMP REGISTERS CH 1 */
    tdGetLong(ts_timestamp_1_bit63_32_ch1, &ts_timestamp1_bit63_32_ch1);
    tdGetLong(ts_timestamp_1_bit31_00_ch1, &ts_timestamp1_bit31_00_ch1);

    tdGetLong(ts_timestamp_2_bit63_32_ch1, &ts_timestamp2_bit63_32_ch1);
    tdGetLong(ts_timestamp_2_bit31_00_ch1, &ts_timestamp2_bit31_00_ch1);

    tdGetLong(ts_timestamp_3_bit63_32_ch1, &ts_timestamp3_bit63_32_ch1);
    tdGetLong(ts_timestamp_3_bit31_00_ch1, &ts_timestamp3_bit31_00_ch1);

    tdGetLong(ts_timestamp_4_bit63_32_ch1, &ts_timestamp4_bit63_32_ch1);
    tdGetLong(ts_timestamp_4_bit31_00_ch1, &ts_timestamp4_bit31_00_ch1);

    tdGetLong(ts_timestamp_5_bit63_32_ch1, &ts_timestamp5_bit63_32_ch1);
    tdGetLong(ts_timestamp_5_bit31_00_ch1, &ts_timestamp5_bit31_00_ch1);
```

```
/* CHANGE BITS IN SECONDS */
timestamp1_ch1 = ts_timestamp1_bit63_32_ch1 * pow(2,32)*(50e-9) +
                 ts_timestamp1_bit31_00_ch1 *(50e-9);
timestamp2_ch1 = ts_timestamp2_bit63_32_ch1 * pow(2,32)*(50e-9) +
                 ts_timestamp2_bit31_00_ch1 *(50e-9);
timestamp3_ch1 = ts_timestamp3_bit63_32_ch1 * pow(2,32)*(50e-9) +
                 ts_timestamp3_bit31_00_ch1 *(50e-9);
timestamp4_ch1 = ts_timestamp4_bit63_32_ch1 * pow(2,32)*(50e-9) +
                 ts_timestamp4_bit31_00_ch1 *(50e-9);
timestamp5_ch1 = ts_timestamp5_bit63_32_ch1 * pow(2,32)*(50e-9) +
                 ts_timestamp5_bit31_00_ch1 *(50e-9);

/* PUT TIME STAMPS IN VECTOR TS */
ts[0] = timestamp1_ch1;
ts[1] = timestamp2_ch1;
ts[2] = timestamp3_ch1;
ts[3] = timestamp4_ch1;
ts[4] = timestamp5_ch1;

/* PUT QUADRATURE COUNTS IN VECTOR XS */
if (ts_counts1_ch1 != b){

     if (ts_counts5_ch1 > xs[0]){
         xs[4] = ts_counts5_ch1 - 0.5;
     }
     else if (ts_counts5_ch1 < xs[0]){
         xs[4] = ts_counts5_ch1 + 0.5;
     }
     else  {
         xs[4] = ts_counts5_ch1;
     }
     if (ts_counts4_ch1 > ts_counts5_ch1) {
         xs[3] = ts_counts4_ch1 - 0.5;
     }
     else if (ts_counts4_ch1 < ts_counts5_ch1) {
         xs[3] = ts_counts4_ch1 + 0.5;
     }
     if (ts_counts3_ch1 > ts_counts4_ch1){
         xs[2] = ts_counts3_ch1 - 0.5;
     }
     else if (ts_counts3_ch1 < ts_counts4_ch1){
         xs[2] = ts_counts3_ch1 + 0.5;
     }
     if (ts_counts2_ch1 > ts_counts3_ch1) {
         xs[1] = ts_counts2_ch1 - 0.5;
     }
     else if (ts_counts3_ch1 < ts_counts4_ch1)  {
         xs[1] = ts_counts4_ch1 + 0.5;
     }
     if (ts_counts1_ch1 > ts_counts2_ch1){
         xs[0] = ts_counts1_ch1 - 0.5;
     }
     else if (ts_counts1_ch1 < ts_counts2_ch1){
         xs[0] = ts_counts1_ch1 + 0.5;
     }

     b = ts_counts1_ch1;

     /* CALCULATE POLYNOMIAL COEFFICIENTS */

     GetCoefficients3orderFit(ts, xs, o, nts, ptr_cf);

     /* DETERMINE POSITION ESTIMATION WITH POLYNOMIAL */

     x = ptr_cf[0]*pow(t,3) + ptr_cf[1]*pow(t,2) + ptr_cf[2]*t + ptr_cf[3];
```

```
            v = 3*ptr_cf[0]*pow(t,2) + 2*ptr_cf[1]*t + ptr_cf[2];


            /* WRITE POSITION INFORMATION TO FILE */
            fprintf(pf, "%f\t%i\t%i\t%f\t%f\t%f\t%f\n",t,ts_counts1_ch2,ts_counts1_ch1,x,v,xs[0],ts[0]);
        }

        else if (ts_counts1_ch1 == b){

            x = ptr_cf[0]*pow(t,3) + ptr_cf[1]*pow(t,2) + ptr_cf[2]*t + ptr_cf[3];

            v = 3*ptr_cf[0]*pow(t,2) + 2*ptr_cf[1]*t + ptr_cf[2];


            /* WRITE POSITION INFORMATION TO FILE */
            fprintf (pf, "%f\t%i\t%i\t%f\t%f\t%f\t%f\n",t,ts_counts1_ch2,ts_counts1_ch1,x,v,0.0,0.0);
       }

    n = abs(ts_counts1_ch1);
    }
    getchar();
    fclose (pf);
    tdClose( );
}
```

C.2 1^{st} Order polynomial fit

```
////////////////////////////////////////////////////////////////////////
////////////// 1ST ORDER POLYNOMIAL FIT ////////////////////////////////
////////////////////////////////////////////////////////////////////////

int GetCoefficients1orderFit(double ts[], double xs[], int o, int nts, double *ptr_cf) {

    /* DECLARATIONS VARIABLES AND ARRAYS */
    double AtA[2][2]= {{0,0},
                       {0,0}};
    double Atb[2]   = {0,0};
    double z[2]     = {0,0};
    double m[2][2]  = {{0,0},
                       {0,0}};
    int i;
    int j;
    int k;
    int n;

    /*Solve A'*A*coeff = A'*b */

    /*1. Calculate A'*A */
    for (i=0; i<=1; i++) {
        for (j=0; j<=1; j++) {
            for (n=0; n<nts; n++) {
                AtA[i][j] = AtA[i][j] + pow(ts[n], ((1+1)-i-j));
            }
        }
    }

    //2. Calculate A'*b
    for (j=0; j<=1; j++) {
        for (n=0; n<nts; n++) {
            Atb[j] = Atb[j] + pow(ts[n], (1-j))* xs[n];
        }
    }

    //3. LU decomposition
```

```
    for (k=0; k<1; k++) {
        for (i=k+1; i<=1; i++) {
            m[i][k] = AtA[i][k]/AtA[k][k];
        }
        for (j=k+1; j<=1; j++) {
            for (i=k+1; i<=1; i++) {
                AtA[i][j] = AtA[i][j] - m[i][k]*AtA[k][j];
            }
        }
    }

    //4. Forward substitution
    z[0] = Atb[0];
    z[1] = Atb[1] - m[1][0]*z[0];

    //5. Backward substitution
    ptr_cf[1] = z[1]/AtA[1][1];
    ptr_cf[0] = (z[0] - AtA[0][1]*ptr_cf[1])/AtA[0][0];

    return 0;
}
```

C.3 2^{nd} Order polynomial fit

```
////////////////////////////////////////////////////////////////////////
////////////// 2ND ORDER POLYNOMIAL FIT ////////////////////////////////
////////////////////////////////////////////////////////////////////////

int GetCoefficients2orderFit(double ts[], double xs[], int o, int nts, double *ptr_cf) {

    //DECLARATIONS VARIABLES AND ARRAYS
    double AtA[3][3]= {{0,0,0},
                       {0,0,0},
                       {0,0,0}};
    double Atb[3]   = {0,0,0};
    double z[3]     = {0,0,0};
    double m[3][3]  = {{0,0,0},
                       {0,0,0},
                       {0,0,0}};
    int i;
    int j;
    int k;
    int n;


    //Solve A'*A*coeff = A'*b

    //1. Calculate A'*A
    for (i=0; i<=2; i++) {
        for (j=0; j<=2; j++) {
            for (n=0; n<nts; n++) {
                AtA[i][j] = AtA[i][j] + pow(ts[n], ((2+2)-i-j));
            }
        }
    }

    //2. Calculate A'*b
    for (j=0; j<=2; j++) {
        for (n=0; n<nts; n++) {
            Atb[j] = Atb[j] + pow(ts[n], (2-j))* xs[n];
        }
    }

    //3. LU decomposition
    for (k=0; k<2; k++) {
```

```
        for (i=k+1; i<=2; i++) {
            m[i][k] = AtA[i][k]/AtA[k][k];
        }
        for (j=k+1; j<=2; j++) {
            for (i=k+1; i<=2; i++) {
                AtA[i][j] = AtA[i][j] - m[i][k]*AtA[k][j];
            }
        }
    }

    //4. Forward substitution
    z[0] = Atb[0];
    z[1] = Atb[1] - m[1][0]*z[0];
    z[2] = Atb[2] - m[2][0]*z[0] - m[2][1]*z[1];

    //5. Backward substitution
    ptr_cf[2] = z[2]/AtA[2][2];
    ptr_cf[1] = (z[1] - AtA[1][2]*ptr_cf[2])/AtA[1][1];
    ptr_cf[0] = (z[0] - AtA[0][2]*ptr_cf[2] - AtA[0][1]*ptr_cf[1])/AtA[0][0];

    return 0;
}
```

C.4 3^{rd} Order polynomial fit

```
////////////////////////////////////////////////////////////////////////
////////////// 3RD ORDER POLYNOMIAL FIT ////////////////////////////////
////////////////////////////////////////////////////////////////////////

int GetCoefficients3orderFit(double ts[], double xs[], int o, int nts, double *ptr_cf) {

    //DECLARATION VARIABLES AND ARRAYS

    double AtA[4][4]= {{0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0}};
    double Atb[4]   = {0,0,0,0};
    double z[4]     = {0,0,0,0};
    double m[4][4]  = {{0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0},
                       {0,0,0,0}};
    int i;
    int j;
    int k;
    int n;

    //STATEMENTS TO CALCULATE COEFFICIENTS
    //Solve A'*A*coeff = A'*b

    //1. Calculate A'*A
    for (i=0; i<=3; i++) {
        for (j=0; j<=3; j++) {
            for (n=0; n<nts; n++) {
                AtA[i][j] = AtA[i][j] + pow(ts[n], ((3+3)-i-j));
            }
        }
    }

    //2. Calculate A'*b
    for (j=0; j<=3; j++) {
        for (n=0; n<nts; n++) {
            Atb[j] = Atb[j] + pow(ts[n], (3-j))* xs[n];
        }
```

```
    }

    //3. LU decomposition
    for (k=0; k<3; k++) {
        for (i=k+1; i<=3; i++) {
            m[i][k] = AtA[i][k]/AtA[k][k];
        }
        for (j=k+1; j<=3; j++) {
            for (i=k+1; i<=3; i++) {
                AtA[i][j] = AtA[i][j] - m[i][k]*AtA[k][j];
            }
        }
    }

    //4. Forward substitution
    z[0] = Atb[0];
    z[1] = Atb[1] - m[1][0]*z[0];
    z[2] = Atb[2] - m[2][0]*z[0] - m[2][1]*z[1];
    z[3] = Atb[3] - m[3][0]*z[0] - m[3][1]*z[1] - m[3][2]*z[2];

    //5. Backward substitution
    ptr_cf[3] = z[3]/AtA[3][3];
    ptr_cf[2] = (z[2] - AtA[2][3]*ptr_cf[3])/AtA[2][2];
    ptr_cf[1] = (z[1] - AtA[1][3]*ptr_cf[3] - AtA[1][2]*ptr_cf[2])/AtA[1][1];
    ptr_cf[0] = (z[0] - AtA[0][3]*ptr_cf[3] - AtA[0][2]*ptr_cf[2] - AtA[0][1]*ptr_cf[1])/AtA[0][0];

    return 0;
}
```

Bibliography

[1] L. Ljung and T. Glad, "Velocity estimation from irregular, noisy position measurements", *IFAC 9th Triennial World Congress, Budapest*, 2:1069-73, 1984.

[2] P.R. Bélanger, "Estimation of angular velocity and acceleration from shaft encoder measurements", *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, pages 585-592, May 1992.

[3] Y. Buchnik and R. Rabinovici, "Speed and position estimation of brushless dc motor in very low speeds", *IEEE voor de rest nog opzoeken*, pages 317-320, 2004.

[4] R.H. Brown, S.C. Schneider and M.G. Mulligan, "Analysis of algorithms for velocity estimation from discrete position versus time data", *IEEE Transactions on industrial electronics*, 39:11-19, 1992.

[5] P.S. Carpenter, R.H. Brown, J.A. Heinen and S.C. Schneider, "On algorithms for velocity estimation using discrete position encoders", *IEEE nog op zoeken*,pages 844-849, 1995.

[6] M.F. Benkhoris and M. Aït-Ahmed, "Discrete speed estimation from a position encoder for motor drives", *6th International Conference on Power Electronics and Variable Speed Drives*,429:283-287, 1996.

[7] S. D'Arco, L. Piegari and R. Rizzo, "An optmised algorithm for velocity estimation method for motor drives", *IEEE Symposium on Diagnostics for Electric Machines, Power Electronics and Drives*, pages 76-80, 2003

[8] F. Janibi-Sharifi, V. Hayward and C.J. Chen, "Discrete-time adaptive windowing for velocity estimation", *IEEE Transactions on control systems technology*, 8:1003-1009, 2000.

[9] N.C. Cheung, "An innovative method to increase the resolution of optical encoders in motion servo systems", *IEEE International Conference on Power Electronics and Drive Systems*, pages 797-802, 1999.

[10] K.K. Tan,H.X. Zhou and T.H. Lee, "New interpolation method for quadrature encoder signals", *Transactions on instrumentation and measurement*, 51:10731079, 2002.

[11] K.Z. Tang, K.K. Tan, T.H. Lee and C.S. Teo, "Neural Network-based correction and interpolation of encoder signals for precision motion control", *IEEE nog op zoeken*, pages 499-504, 2004.

[12] K.Z. Tang, K.K. Tan, "Adaptive online correction and interpolation of quadrature encoder signals using radial basis functions", *IEEE Transaction on control systems technology*, 13:370-377, 2005.

[13] C. Vottis "Extracting more accurate position and velocity information from optical incremental encoders", *Eindverslagen Stan Akkermans Instituut*, 069, 2003.

[14] F.C. van Nijmweegen and J.C.L. van de Griendt "User manual TUeDACS/1 USB MicroGiant", *TU/e Department of physics, Laboratory Automation Group*, 2005.

[15] M.T. Heath "Scientific Computing" *University of Illinois, Urbana-Champaign*, 2002