UNIVERSITY OF PADUA

FACULTY OF ENGINEERING

Finished on the day March 25, 2011 using $\LaTeX\,2_\varepsilon$

UNIVERSITY COLLEGE CORK

—

Department of Electrical and Electronic Engineering

—

MASTER THESIS IN AUTOMATION ENGINEERING

VELOCITY ESTIMATION AND MOTION CONTROL USING MEMS ACCELEROMETER

Advisor: Dr. Luca Schenato

Co-advisor: Dr. Richard Kavanagh

Author: LUCA FARDIN

ACADEMIC YEAR 2010-2011

to my family...

“ Everybody knows that something is impossible, until it reaches a fool who does not know and invents. ”

ALBERT EINSTEIN, 1879-1955

Contents

Declaration

This report was written entirely by the author, except where stated otherwise. The source of any material not created by the author has been clearly referenced. The work described in this report was conducted by the author except where stated otherwise.

Luca Fardin, 20 of January 2011

Acknowledgements

I feel that thanks are due, especially to Dr. Richard Kavanagh for his suggestions and assistance in developing this project. I also wish to thank Dr. Luca Schenato, who dealt with the operation from our Italian headquarters.

Sincere thanks to Dr. Gerard O'Donovan of Moog Ltd. for his assistance in adapting the Moog drive, and for his help throughout. We are also grateful to Mr. Maurice O'Connor for his help with hardware adjustment and sensor set-up, and to Mr. Michael O'Shea for modifying the mechanical rig.

Last but not least, heartfelt thanks to my parents, who are my point of reference, my sister Monica, and Arianna, who has patiently stayed by my side and supported my efforts over the years.

My gratitude to everyone who has offered me friendship and solidarity.

Sommario

Lo scopo di questo progetto è di studiare le prestazioni del kinematic Kalman filter (KKF) utilizzando un accelerometro al fine di stimare la velocità. Il progetto si è basato su quello coniato da Jeon Soo e Masayoshi Tomizuka [1], che mira ad analizzare i tradizionali metodi di stima della velocità e confrontarli con il KKF. Al fine di misurare l'accelerazione angolare con un metodo affidabile e conveniente, è stato utilizzato un Micro-Electro-Mechanical System MEMS accelerometro. Questo progetto dimostra che il kinematic Kalman filter presenta prestazioni elevate anche con l'utilizzo di un encoder a bassa risoluzione, in quanto è insensibile alla modellizzazione dell'incertezze e dalle variazioni dei parametri.

The purpose of this project is to investigate the performance of the kinematic Kalman filter (KKF) by using an accelerometer in order to estimate the velocity. The project was based upon the one coined by Soo Jeon and Masayoshi Tomizuka [1], which aims to analyze conventional velocity estimation methods and compare them to the KKF. In order to measure angular acceleration with a reliable and cost-effective method a Micro-Electro-Mechanical System MEMS accelerometers was used. This project shows that the kinematic Kalman filter can perform very well even with a low-resolution encoder, as it is insensitive to modeling uncertainties and parameter variations.

Chapter 1

Introduction

1.1 Purpose

The purpose of this work is to investigate the performance of the kinematic Kalman filter (KKF). The project is based upon the one coined by Soo Jeon and Masayoshi Tomizuka [1], which aims to analyze conventional velocity estimation methods and compare them with the KKF. In order to measure angular acceleration with a reliable and cost-effective method, Micro-Electro-Mechanical System MEMS accelerometers have been used.

This research was conducted to obtain very accurate velocity information, as it often required in control laws for mechanical systems. Usually the velocity is estimated from the encoder position because a pulse encoder is the most typical sensor in motion control. The simplest method is based on the difference of successive encoder counts, or a better approach is to time the interval between two consecutive encoder pulses at the expense of a large phase lag. At high speeds, these methods may provide a relatively accurate estimate of velocity, but at low speeds the estimate becomes highly unreliable. Another method is to use a high-resolution encoder, which can provide an accurate velocity estimate at the expense of higher implementation costs.

In the project, the estimated velocity has been implemented based on the model-based state estimation theory in order to compare the results with the KKF. In the model-based approaches, model parameters and external disturbance must be accurately known for the estimate of velocity to be accurate, which is very difficult in reality. Using accelerometers for velocity estimation, no model parameter is needed if we use a kinematic model relating to the position and use the acceleration measurement as input to set up a kinematic Kalman filter (KKF). For this reason, the most attractive feature of KKF is that it is insensitive to modeling uncertainties and parameter variations.

1.2 Micro Electro-Mechanical System Devices

MEMS Micro Electro-Mechanical Systems is the technology of very small mechanical devices driven by electricity.

MEMS are made up of components between 1 to 100 micrometers in size and MEMS devices generally range in size from 20 micrometers to a millimeter.

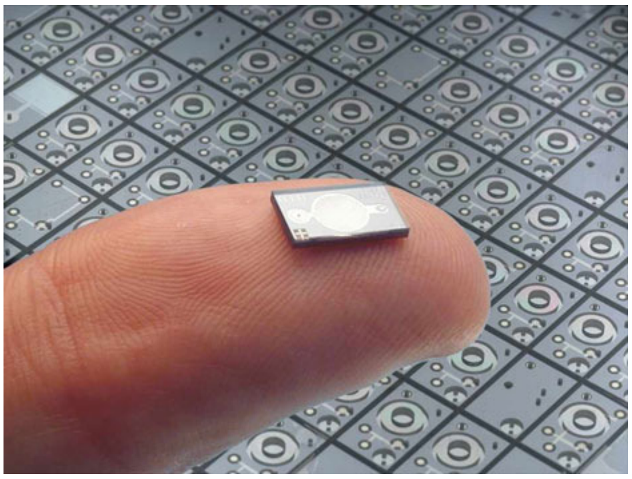


Figure 1.1: Example MEMS nanopump.

They usually consist of a central unit that processes data, the microprocessor and several components that interact with the outside such as microsensors. At these size scales, the standard constructs of classical physics are not always useful. Because of the large surface area to volume ratio of MEMS, surface effects such as electrostatics and wetting dominate volume effects such as inertia or thermal mass. MEMS became practical once they could be fabricated using modified semiconductor device fabrication technologies, normally used to make electronics. These include moulding and plating, wet etching and dry etching, electro discharge machining, and other technologies capable of manufacturing small devices.

The types of MEMS devices can vary from relatively simple structures having no moving elements, to extremely complex electromechanical systems with multiple moving elements under the control of integrated microelectronics. The one main criterion of MEMS is that there are at least some elements having some sort of mechanical functionality whether or not these elements can move.

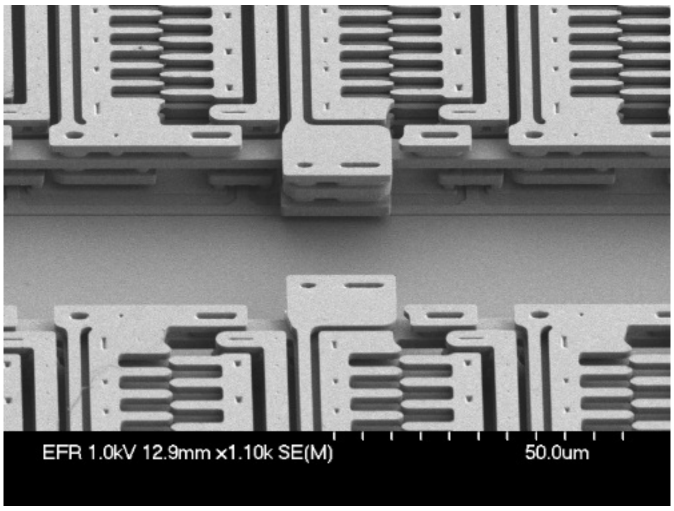


Figure 1.2: Detail view of MEMS electrostatic actuator.

Commercial applications with MEMS devices may be utilized, for example:

- Inkjet printers, which use piezoelectrics or thermal bubble ejection to deposit ink on paper;
- Accelerometers in modern cars for a large number of purposes including airbag deployment in collisions;
- Accelerometers in consumer electronics devices such as game controllers (Nintendo Wii), personal media players / cell phones (Apple iPhone, various

Nokia mobile phone models, various HTC PDA models) and a number of Digital Cameras (various Canon Digital IXUS models). Also used in PCs to park the hard disk head when free-fall is detected, to prevent damage and data loss;

- MEMS gyroscopes used in modern cars and other applications to detect yaw; e.g., to deploy a roll over bar or trigger dynamic stability control;
- Silicon pressure sensors e.g., car tire pressure sensors, and disposable blood pressure sensors;
- Bio-MEMS applications in medical and health related technologies from Lab-On-Chip to MicroTotalAnalysis (biosensor, chemosensor).

In other words, the development of linear and rotational MEMS accelerometers has been heavily promoted by the automotive manufacturing and computer hardware industries. Recently, MEMS devices have had quite an impact on the medical industry, using procedures related to the measuring of blood pressure and other procedures for biomedical applications.

1.3 Estimation velocity

Due to the increasing use of mechanical and automated systems for many industrial applications, the demand of performance for such mechanical systems has increased. In many cases, the high performance of accurate positioning and tracking of trajectories in mechanical systems are required.

Therefore, control laws for these systems often require very accurate velocity information. Usually, the velocity is estimated from the encoder position only because a pulse encoder is the most typical sensor in motion control. In this case, the simplest method to estimate the velocity is based on the difference of successive encoder counts.

On the market there are many types of encoders, for example the incremental encoders provide a specific number of equally spaced pulses per revolution (PPR) of linear motion. A single channel output is used for applications where sensing the direction of movement is not important. Where direction sensing is required, quadrature output is used, with two channels 90 electrical degrees out of phase. Circuitry determines direction of movement based on the phase relationship between them.

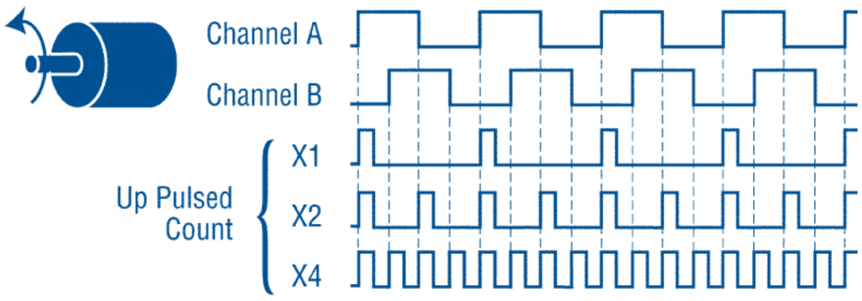


Figure 1.3: Incremental encoder counts.

When more resolution is needed, it's possible for the counter to count the leading and trailing edges of the pulse train from one channel, which doubles (x2) the number of pulses counted for one rotation of motion. Counting both leading and trailing edges of both channels will give (x4) resolution. To determine position, its pulses must be accumulated by a counter. The count is subject to loss during a power interruption or corruption by electrical transients. Some incremental encoders also produce another signal known as the "index" or "Z channel". This signal, produced once per revolution of a shaft encoder or at precisely-known points on a linear scale, is often used to locate a specific position.

The measurement resolution of the velocity is given by $\delta\omega = \delta\vartheta/T_s$, where $\delta\vartheta$ is the resolution of the encoder and T_s is usually the sampling time. $\delta\omega$ is called the velocity resolution and T_s the measurement delay in this case. If higher velocity resolution is required using the same encoder, the measurement time delay increases because the product of the two quantities is fixed by the resolution of the encoder $\delta\vartheta$. Therefore, the resolution of the velocity estimation becomes directly proportional to the resolution of the encoder in this method. This method may provide a relatively accurate estimate of the velocity at high speeds, but at low speeds the estimate becomes highly unreliable. At extremely low speeds, a better approach is to time the interval between two consecutive encoder pulses at the expense of a large phase lag.

High-resolution encoder can provide a more accurate velocity estimate, but it greatly increases the implementation cost if we want very high accuracy at low speeds. This is the reason why some of high-precision motion control systems use encoders with a resolution much higher than necessary to satisfy the accuracy requirement for positioning.

The velocity can be estimated using model-based state estimation theory. These approaches, the model parameters and external disturbances must be accurately known for the estimate of velocity to be accurate, which is very difficult to have in reality.

The model-based speed observers [2], [3] make the velocity estimate robust using disturbance observers. In this way, a machine drive technique using novel estimation strategy for the very low-speed operation to estimate both the instantaneous speed and disturbance load torque was proposed. In the proposed algorithm, Kalman filter was incorporated to estimate both the motor speed and the disturbance torque. The effects of parameter variations were discussed.

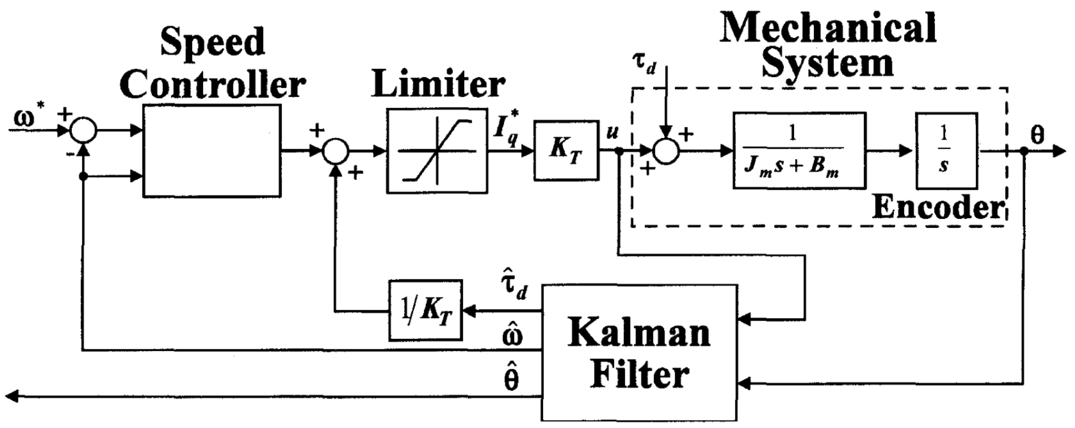


Figure 1.4: The block diagiram of speed controller including speed and disturbance observer using Kalman filter. Figure found in [3].

Another way, the speed averaging was used to improve the performance at very low speeds [4]. However, these methods require accurate system parameters such as inertia and friction. Furthermore, the disturbance observer cannot successfully track fast changing or discontinuous disturbances such as cogging force and stiction.

Therefore, these methods are not so robust in applications such as robot manipulators and time varying loads. So it is not possible to use these methods to wide range of working conditions while maintaining robustness and accuracy.

The use of accelerometers in motion control has been suggested for some time. Some results have been reported for linear motors [5] and hard disk drive systems [6].

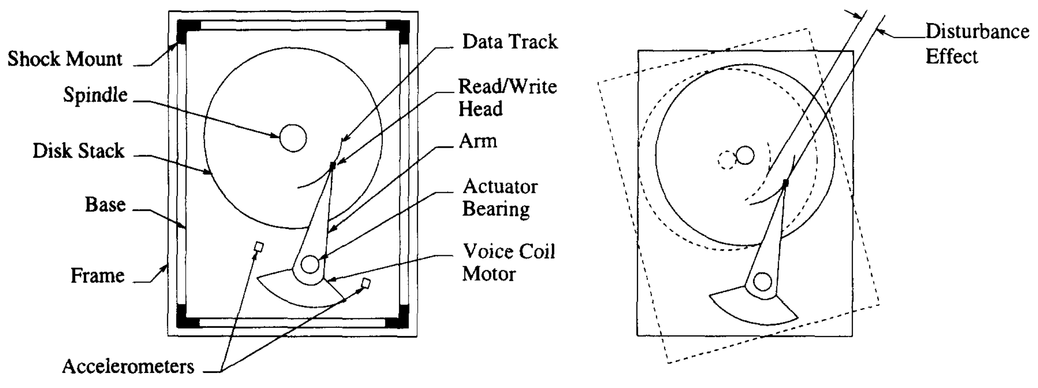


Figure 1.5: Disk drive components and effect of disturbance on track misregistration. Figure found in [6].

For example, in this application the accelerometers were used to measure the motion of the drive, and then feed this information forward to the actuator controller to coordinate the read/write head position with the desired track position.

The significant benefits using this additional sensor is that the robustness to model parameters in estimating state variables increases. In fact, for velocity estimation, no model parameter is needed if we use a kinematic model relating the position to the acceleration and use the acceleration measurement as an input to set up a kinematic Kalman filter (KKF).

Chapter 2

Relevant theory

2.1 MEMS Accelerometer

The information provided in this section can be found in [7] and [8].

The MEMS device chosen for this project is the ADXL210E Analog Device™unit. The ADXL210E is a low-cost, low-power, complete 2-axis accelerometer with a digital output, all on a single monolithic IC. The ADXL210E will measure accelerations with a full-scale range of ±10 g. The ADXL210E can measure both dynamic acceleration and static acceleration (e.g. gravity).The outputs are analog voltage or digital signals whose duty cycles are proportional to acceleration. The duty cycle outputs can be directly measured by a microprocessor counter without an A/D converter. For these reasons and for its ultra-small size this type of device has been chosen. Another reason is that the same device had been employed by Jeon and Tomizuka in their experiment [1].

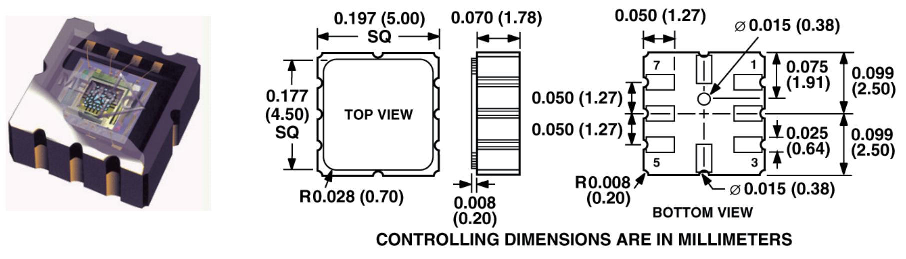


Figure 2.1: MEMS accelerometer ADXL210. Figures found in [8] and [9].

The MEMS acceleration sensor is based upon a change in an electrical characteristic such as capacitance. There is a fixed anchored section and a moveable mass held by Polysilicon springs. These springs suspend the sensor structure over the substrate and also provide resistance against linear acceleration forces. The structure of the sensor is called Proof Mass.

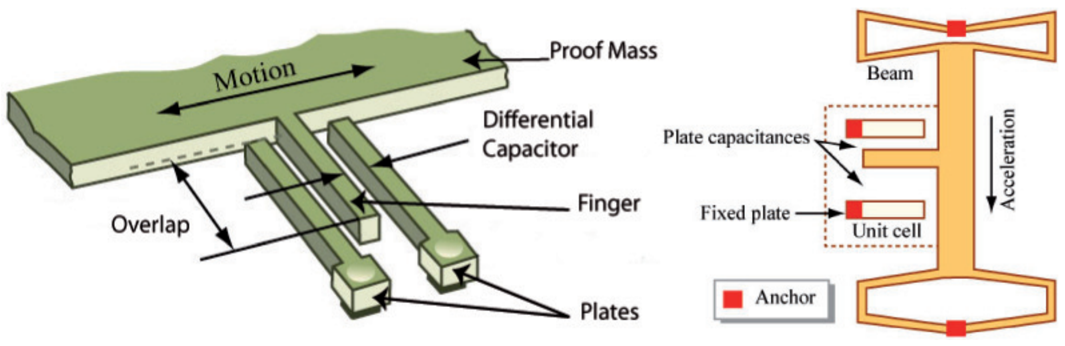


Figure 2.2: Left: Section of MEMS accelerometer sensor. Right: Simplified view of the MEMS accelerometer under acceleration.

When the device is accelerated the moveable section is shifted in relation to the fixed section in the same way that a passenger is thrown forward when a car brakes suddenly. Consequently, the fingers which are fixed to the Proof Mass will move relative to the plats and the finger will create a differential capacitor, whose capacitance alters proportionally to the motion of the Proof Mass. The size of the movement depends not only on how large the force is but also on the mass of the moveable section and the strength of the springs. These are selected to make the accelerometer sensitive to a particular size of acceleration. This difference of capacitance is detected and amplified to produce a voltage proportional to the linear acceleration. Figure 2.2 shows the micro structure sensor and the principle of operation. This structure is repeated numerously on all four sides of the Proof Mass in order to measure the linear acceleration in perpendicular X and Y directions. The dimensions of the structure are of the order of microns.

2.2 Signal transmission: Slip Rings

A multi-element Mercury Slip-Ring was used in this project to transfer a signal between a rotating mass and a fixed body. Using this method, the Slip Ring can also supply constant DC voltage to the MEMS.

A Slip Ring is a method for making an electrical connection through a rotating assembly. Slip rings are commonly found in electric motors, electrical generators for AC systems and alternators and in the packaging machinery, cable reels, and wind turbines.
One of the two rings is connected to one end of the field winding and other one to the other end of the field winding. A common brass Slip Ring consists of a conductive circle or band mounted on a shaft and insulated from it. Electrical connections are made to the ring from the rotating part of the system, such as the rotor of a generator. Fixed contacts or brushes run in contact with the ring, transferring electrical power or signals to the exterior static part of the system.

Mercury-wetted Slip Rings are noted for their low resistance and stable connection and use a different principle which replaces the sliding brush contact with a pool of liquid metal molecularly bonded to the contacts. During rotation the liquid metal maintains the electrical connection between the stationary and rotating contacts.

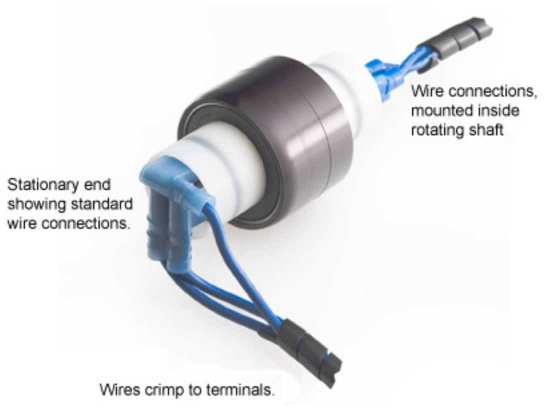

Wire connections, mounted inside rotating shaft
Stationary end showing standard wire connections.
Wires crimp to terminals.

Figure 2.3: Mercotac Mercury Slip Rings.

As described in [7], Mercury Slip Ring was chosen for the project as common brass Slip Rings create considerable noise and also have very high wear. Moreover, brass Slip Rings have a much shorter life span than Mercury Slip Rings. In fact, the lifetime of a brass Slip Ring is merely into the millions of revolution and that of a mercury Slip Rings is more than a billion revolutions. Another advantage of Mercury Slip Rings is that they have almost zero impedance on the dynamics of the system, and therefore will not introduce any noise.

Chapter 3

Experimental apparatus

The experimental apparatus consists of the following main parts:

- Brushless servomotor Moog G400 Series with resolver;
- Moog DS2100CAN Digital Controller Driver;
- MEMS ADXL210JE Accelerometers Devices attached on the outside of the disk;
- Digital Signal Processing (DSP): dSPACE DS1102 Floating-Point Controller Board;
- PC Pentium 1, 32MB of RAM, 1GB of ROM with Windows 98 and DSP processor;
- PC Pentium 4, CPU 3 GHz, 1GB of RAM with Windows XP and MOOG Motor GUI software;
- Incremental encoder from *Leine & Linde* with 4096 PPR;
- Anologue Signal Processing: Vero-Board Circuit and RC Lowpass Filter and Decoupling Capacitors;
- Tektronix TDS 2014 Four channel Digital Storage Oscilloscope 100MHz 1GS/s;
- 4 Conductor Mercury Slip-Ring *Mercotac Model 430*;

- 24V Controller supply and 5V MEMS accelerometer supply;
- Hoodwin Acceleration Sensor, not required in the project.

Figure 3.20 shows an overview of the system with all its components.

The main parts of the apparatus and their characteristics are analysed in subsequent sections.

3.1 Brushless servomotor Moog G400 Series

The characteristics and values in this section can be found in the datasheet [10].

Moog's G400 Series motors are electronically commutated synchronous AC motors with permanent magnet field excitation. The motor model used in the project is G424_414A. Figure 3.1 shows a diagram of the motor with resolver.

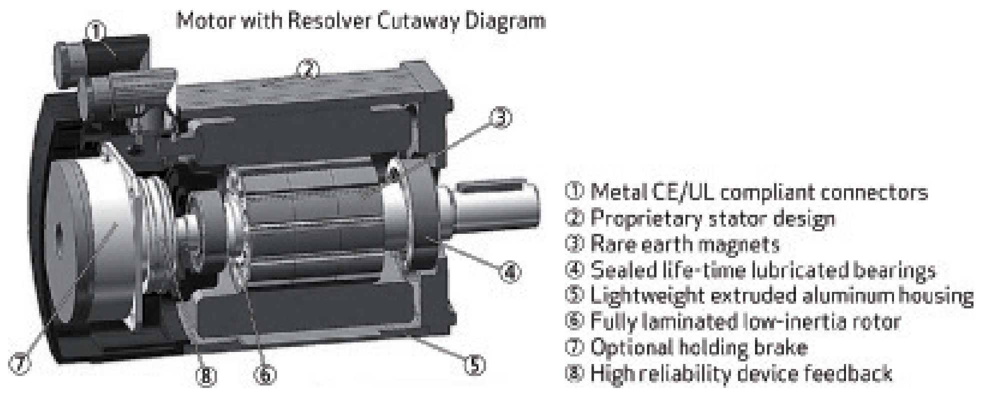

Motor with Resolver Cutaway Diagram
① Metal CE/UL compliant connectors
② Proprietary stator design
③ Rare earth magnets
④ Sealed life-time lubricated bearings
⑤ Lightweight extruded aluminum housing
⑥ Fully laminated low-inertia rotor
⑦ Optional holding brake
⑧ High reliability device feedback

Figure 3.1: Motor with Resolver diagram. Figure found in [10].

One benefit of using this motor is that it can be controlled by DS2100 controller and interfaced with a personal computer. In fact, this type of device uses a graphical user interface (GUI), called *Win Drive*, as described in the next section.
Its main characteristics and nominal values with sinusoidal drive are:

Type	Symbol	Value	Units
Nominal Torque	M_o	2.6	Nm
Nominal speed	n_N	5500	rpm
Maximum speed	n_{max}	8000	rpm
Nominal current	I_o	4.8	Arms
Peak current	I_p	15	Arms
Output power	P_N	0.95	kW
Torque constant	k_t	0.56	Nm/Arms
Voltage constant	k_e	34.2	Vrms/krpm
Rotor inertia with resolver	J	2.09	kg cm^2
Winding resistance at 25°C	R	2.6	Ohm
Winding inductance (phase to phase)	L	5.8	mH
Motor pole count	n_p	12	

Table 3.1: Performance specification for motor model G424_414A. Data at 25°C.

3.2 Moog DS2100CAN Digital Controller Driver

This section contains information found in [11].

The DS2100 provides full digital control of brushless servomotors and utilizes a microprocessor to deliver significantly increased current, velocity and position performance. It also provides a full range of interfaces to motor, feedback devices and higher-level controllers. The DS2100 provides high performance loop closure via full digital control. It has torque, velocity and position control capability according to the following main features (100 μsec sampling time):

- torque loop based on Space Vector Modulation, observers and PI control;
- velocity loop based on a classical PI configuration with programmable digital filters (Low-pass,High-pass,Band-pass, Band-stop);
- position loop based on classical PI or time optimal control.

Figure 3.2 shows the DS2100 digital controller and its interface.

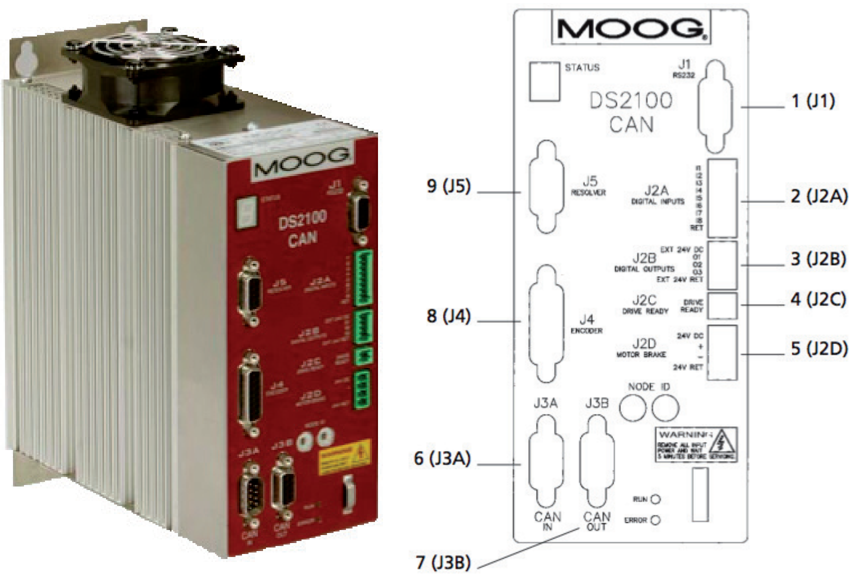
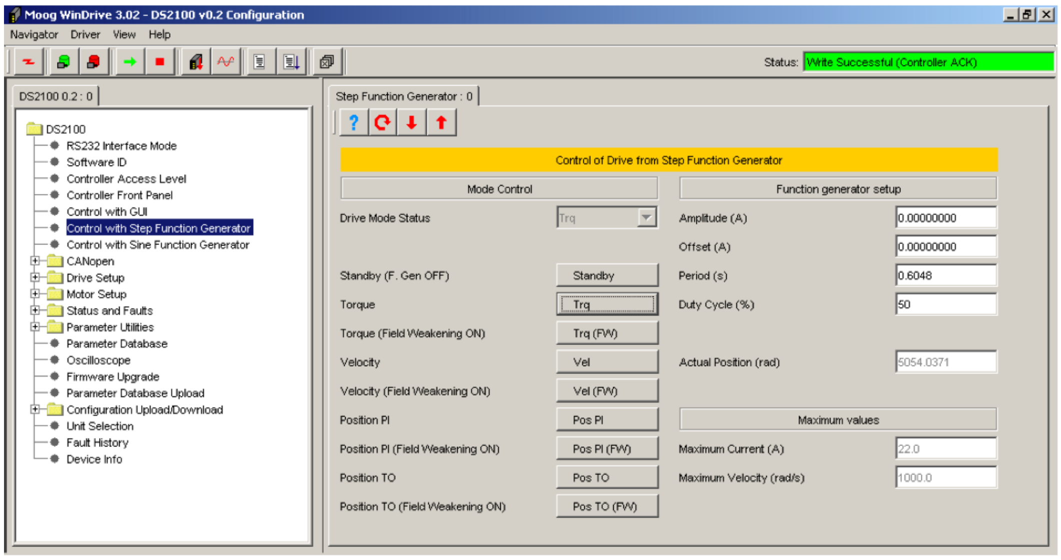


Figure 3.2: DS2100 Digital Controller. Interfaces:1)RS232 connector; 2) digital inputs; 3) digital outputs; 4) drive ready; 5) motor brake control; 6) CAN input; 7) CAN output; 8) encoder input; 9) resolver input.

The drive programming was performed by use of a management software program called *Win Drive*, Graphical User Interface (GUI), based on Java. The version of the software used in the project was 2.0 provided by Moog engineers. This software requires the use of the Windows XP platform. The specifications of the computer used for motor GUI are: PC Pentium 4, CPU 3 GHz, 1GB of RAM. Figure 3.3 shows an illustration of the Graphical User Interface.
Access to the following functions will be available:

- Continuous Serial Communications status monitoring;
- Downloading and uploading files related to drive parameters;
- Real time DS2100 virtual front panel;
- Downloading of firmware;
- Modification and adjustment of drive parameters;
- Real time oscilloscope function.



Status: Write Successful (Controller ACK)

Drive Mode Status: Trq

Amplitude (A): 0.00000000

Offset (A): 0.00000000

Period (s): 0.6048

Duty Cycle (%): 50

Actual Position (rad): 5054.0371

Maximum Current (A): 22.0

Maximum Velocity (rad/s): 1000.0

Figure 3.3: Screenshot of Motor GUI. Figure found in [11].

Moreover the GUI allows direct data acquisition for position velocity and other signals, but this data is accessible only by digital means and is consequently limited to three data bits per byte of information.

Using information provided by a Moog engineer it has been possible to access a current loop with the driver. The engineer showed how to bypass the standard digital input to the drive and apply an analog required torque voltage. In this way, an analog output from the dSPACE system can be connected to the DS2100 and provide an analogue torque input to the driver.

Figure 6.25 shows the schematic of the DS2100 driver and the two points where input voltage is installed. Information was provided by Moog.
To use the current-loop, the GUI must be set in a command reference and analog torque mode. To do this, the parameter "modreq" must be set to 8209, which means torque mode and use of the ADC command.
Particular attention is paid to the fact that the values of input voltage vary over a range of 0-4.85V and notes the calculations in the software assumes that the signal is biased at 2.44V.

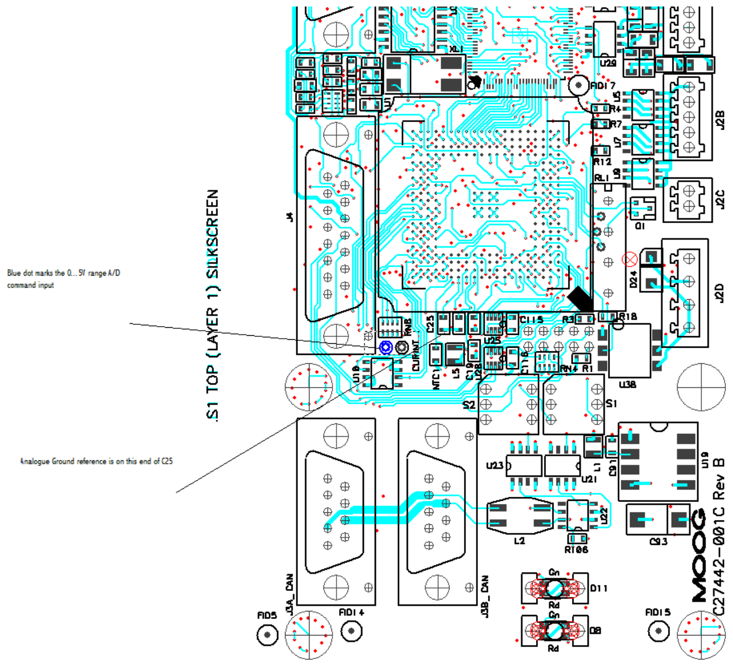

Blue dot marks the 0...5V range A/D command input
Analogue Ground reference is on this end of C25
S1 TOP (LAYER 1) SILKSCREEN
MOOG
C27442-001C Rev B

Figure 3.4: DS2100, digital card layout.

The A/D input value can be viewed on adccmd_g parameter in the database tab of the GUI.
After some tests on the system, it was found that the voltage input into the driver and the current supplied to the motor have the following relationship:

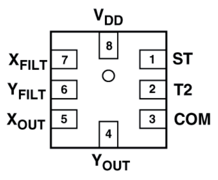
- 0 V = -32704 adccmd_g increments (maps to +imax amps command to the current loop);
- 2.425 V = 0 adccmd_g increments (maps to 0 amps command to the current loop);
- 4.85 V = +32767 adccmd_g increments (maps to -imax amps command to the current loop).

Therefore, it needs to supply a 2.425V bias on the input to achieve the command 0 A current condition in the loop.

3.3 MEMS Accelerometers characterization

This section presents in detail the characteristics of the MEMS device used in the project. Some informations were found in [7], [8] and [12].

The MEMS ADXL210E device used in the project is an 8-pin device. The configuration of the pins in the device is shown in Figure 3.5. Pin 1 (ST) is a Self Test Pin and would only be useful if needed to decipher whether the device is functioning correctly. Pin 2 (T2) is used to set the period of the Duty Cycle. The power to the device is supplied through Pin 8 (VDD) and Pin 3 (COM), 5 V. Pin 4 (Y_{OUT}) and Pin 5 (X_{OUT}) provide a digital output signal and Pin 6 (Y_{FILT}) and Pin 7 (X_{FILT}) give an analog output signal. As written in [8] it is



Pin No.	Mnemonic	Description
1	ST	Self-Test
2	T2	Connect R_{SET} to Set T2 Period
3	COM	Common
4	Y_{OUT}	Y-Channel Duty Cycle Output
5	X_{OUT}	X-Channel Duty Cycle Output
6	Y_{FILT}	Y-Channel Filter Pin
7	X_{FILT}	X-Channel Filter Pin
8	V_{DD}	3 V to 5.25 V

Figure 3.5: Pin Configuration. Figure found in [8].

recommended that a 0.1μF capacitor should be attached across V_{DD} and COM so as to avoid power supply coupling. A surface mount resistor was soldered between two of the MEMS pins, connecting a single resistor between Pin 2 and ground, in order to set the Duty Cycle Measurement (DCM) on a period equal to 1 ms using a resistor of 125 kΩ. The Table in Figure 3.6 shows the relationship between the values of this resistor R_{SET} and the time T2, as well as Counts per g.

Using these devices with dual-axis allows simultaneous measurement in two perpendicular directions, thus allowing acceleration to be sensed on a multi directional rotating sphere. However, in this project only one signal was detected, the axis tangential to the disk. In fact, the axis of rotation of the system was perpen-

T2 (ms)	R_{SET} (kΩ)	ADXL210E Sample Rate	Counter-Clock Rate (MHz)	Counts per T2 Cycle	Counts per *g*	Resolution (m*g*)
1.0	124	1000	2.0	2000	80	12.50
1.0	124	1000	1.0	1000	40	25.00
1.0	124	1000	0.5	500	20	50.00
5.0	625	200	2.0	10000	400	2.50
5.0	625	200	1.0	5000	200	5.00
5.0	625	200	0.5	2500	100	10.00
10.0	1250	100	2.0	20000	800	1.25
10.0	1250	100	1.0	10000	400	2.50
10.0	1250	100	0.5	5000	200	5.00

Figure 3.6: Trade-Offs Between Microcontroller Counter Rate, T2 Period; and Resolution of Duty Cycle Modulatior. The Table can be found in [8].

dicular to gravity,i.e. the MEMS devices were orientated in order to experience the only tangential acceleration and therefore they would ignore centripetal/centrifugal force. Figure 3.7 shows the arrangement of the MEMS accelerometers. Figure 3.8 Left shows the gravitational effect on the MEMS device.

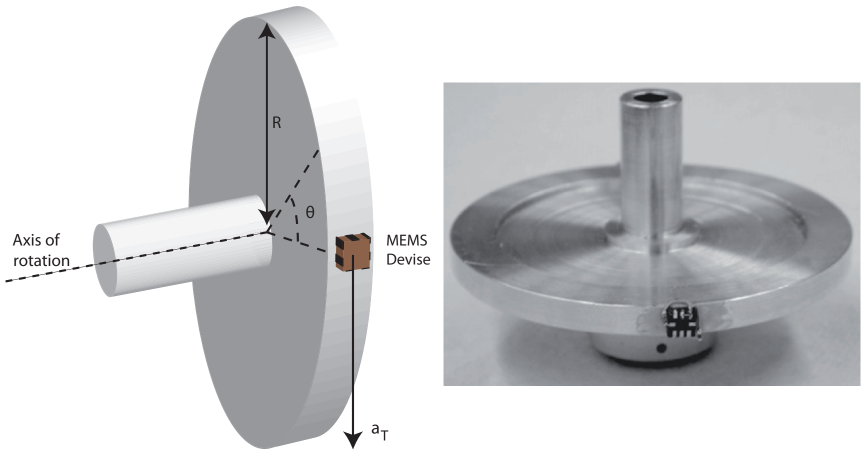


Figure 3.7: MEMS Accelerometers attachment.

The radius of the disk to which the devices are attached is equal to $R = 5.5$ *cm*. At a constant speed, the output signal of the accelerometer is a sinusoidal wave that oscillates around 0 g Offset. In fact, MEMS accelerometers have a fundamental DC Offset for voltage output defined in the datasheet and shown in the following equation:

$$0\text{g Offset} = \frac{V_{dd}}{2} \tag{3.1}$$

Another specification is the sensitivity of MEMS, which is calculated using the following equation:

$$Sensitivity \ = \ 20mV \times V_{dd} \ per \ g \tag{3.2}$$

Therefore, using the voltage supply set to a DC value of 5V, the ideal Zero Offset is 2.5V and the sensitivity is 100mV/g. Then, at a constant speed, the period T of oscillation corresponds to one revolution of the disk. Over this period, the sensor has experienced the full range of gravity.On theory, the output of the sensor then varies between 2.6V and 2.4V. The output signal from a single accelerometer is shown in Figure 3.8 Right.

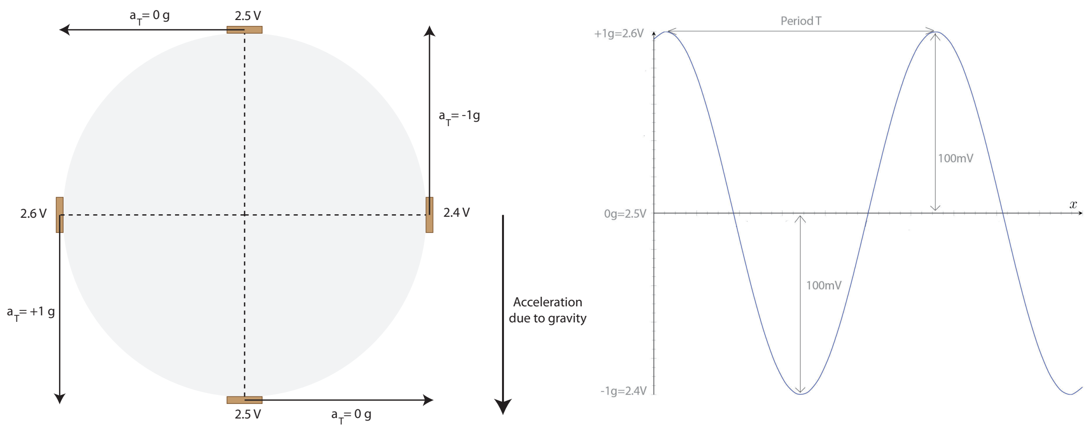


Figure 3.8: Left: Gravitational Effects on the MEMS Device. Right: Ideal MEMS Output Signal.

To avoid this gravitational influence, a second MEMS device is attached 180 degrees out of phase to the first,which can be combined and averaged. With their average the result is a "DC" output signal, approximately equal to zero offset voltage for constant speeds. Therefore, the change of the "DC" signal is correlated with the change of the actual linear acceleration and so the MEMS devices ignore the gravitational effects.

With these ideal assumptions, a relationship can be seen between linear acceleration and the output voltage of the devices. In fact, a variation of 1 mV of the output signal corresponds to a linear acceleration equal to 0.098 m/s^2. Then the angular acceleration can be calculated by the following relationship:

$$\alpha = \frac{a_T}{R} = \frac{0.098}{0.055} = 1.7818 \quad \left[\frac{rad}{\sec^2}\right] \tag{3.3}$$

Therefore, with these ideal assumptions, the sensitivity of the angular acceleration is 178.18 $\left[\frac{rad}{\sec^2}\right]$ for a change in voltage of 100 mV.
Figure 3.9 represents the manufacturer's yield performance for the ADXL210E, regarding axis X used in the project.

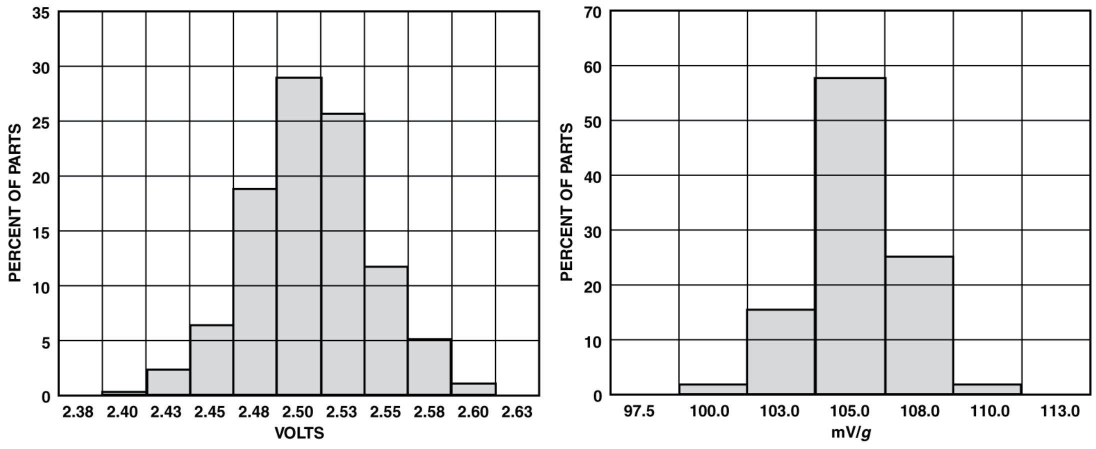


Figure 3.9: VDD = 5 V. Left: X-Axis Zero-g Bias Distribution at X_{FILT}. Right: X-Axis Sensitivity Distribution of X_{FILT}. Figure found in [8].

The real sensitivity of MEMS sensors was calculated in Chapter 9.2 in [7]. These were slightly different from the ideal signals: in fact the averaged signal of the two devices has a sensitivity of 97.5 mV/g and 97.65mV/g in the respective X and Y axes. Using these values shows that a variation of 1 mV corresponds to a linear acceleration of 0.1005 m/s^2, giving an angular acceleration $\alpha = 1.8275\ rad/s^2$.

3.3.1 MEMS Devices Error

As was noted in [7], incorrect placement of MEMS devices on the outer surface of the disk can cause errors in the output signal. Therefore, there may be an error in the measured acceleration signal.

As will be noticed in the following sections, the Zero Offset of the output signal of the accelerometers was not constant, but rather changed as a function of velocity. This change is due to an error in placement of the devices, because the axis of the sensor may not be perpendicular to the radius of the disc and therefore is not just measuring tangential linear acceleration. In fact, the measured signal is also influenced by the centripetal force.

The relationship between centripetal force and centripetal acceleration is given by the equations in 3.4.

$$\begin{aligned} \boldsymbol{F_c} &= -m\boldsymbol{a_c} \\ \boldsymbol{a_c} &= \boldsymbol{\omega} \times (\boldsymbol{\omega} \times \boldsymbol{r}) = -\omega^2 r \boldsymbol{e_r} \end{aligned} \tag{3.4}$$

The centripetal force F_c is always directed toward the centre of the circle, in the direction of the axis of rotation, e_r, and is equal to the mass of the object, m, times the centripetal acceleration, a_c. The effect of a slight misalignment of the devices attached to the disk is shown in Figure 3.10.

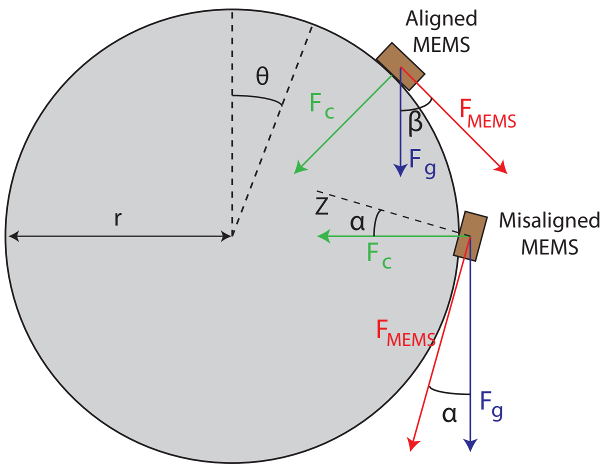


Figure 3.10: Misalignment of the MEMS device.

If the sensors are perfectly placed on the disk surface, the sensor axis Z is parallel to the direction of the centripetal force. In this way, the sensor measures only the force of gravity at some point in a revolution when the disk is rotating at a constant velocity ω. Figure 3.10 shows a misalignment of angle α between the Z-axis sensor and the centripetal force F_c, which also corresponds a misalignment between the X-axis sensor, F_{MEMS}, and the plane of gravitational force, F_g. This is shown when the sensor is in the $+1$g position, i.e. when the angular position θ is 90 degrees. The angle between the X-axis sensor and the plane of gravity is called β and varies with respect to θ. Therefore, at the $+1$g position, α and β are equal. The following equation

$$F_{MEMS} = F_g \cos(\alpha) \pm F_c \sin(\alpha) \tag{3.5}$$

describes the force experienced by device, F_{MEMS}, in relation to the gravitational force, F_g, and the centripetal force, F_c. The following equation describes the force experienced by the device F_{MEMS} in relation to the gravitational force F_g and the centripetal force F_c. The sign of the centripetal force depending on whether the axis of the sensor is orientated towards or away from the centre of the disk. In the case of Figure 3.10, this force is positive relative to the sensor. The acceleration experienced by the sensor is:

$$a_{MEMS} = a_T \cos(\alpha) \pm a_c \sin(\alpha) \tag{3.6}$$

The F_g is based on the tangential acceleration a_T, which depends on the component of the gravity parallel to the tangent, $gCos(\beta)$. Therefore, using the relationships in 3.4, with constant mass, the equation 3.6 can be simplified with the following equation:

$$a_{MEMS} = g \cos(\beta) \cos(\alpha) \pm \omega^2 r \sin(\alpha) \tag{3.7}$$

Therefore, if the device is perfectly aligned, then $\alpha = 0$ degrees, the component of centripetal force is cancelled and $a_{MEMS} = g \cos(\beta)$.

For example, if the sensor has a misalignment angle of 1 degrees towards the centre, and the motor is moving at a constant speed of 10 $[rad/sec]$. The acceleration measured is:

$$a_{MEMS} = 9.8 \cos^2(1) + (10)^2(0.055) \sin(1) = 9.8930 \text{ [m/s}^2\text{]} \tag{3.8}$$

when the angle θ is equal to 90 degrees. Therefore, the acceleration measured has an error equal to 0.0930 $[m/s^2]$ from the real value of acceleration. If the motor is moving at a constant speed of 100 $[rad/sec]$ the acceleration measured is:

$$a_{MEMS} = 9.8\cos^2(1) + (100)^2(0.055)\sin(1) = 19.3958\ [\mathrm{m/s^2}] \tag{3.9}$$

accordingly giving an error of 9.5958 $[m/s^2]$ from the real value of acceleration.

This example is described in order to understand the importance of having devices perfectly aligned. In fact, the error that is caused by misalignment grows linearly with increasing speed, making the sensor worthless.

3.4 Analogue Signal Processing

This section contains information found in [7].

The project implemented in [7] used the analogue signal output from the acceleration sensors. Although ADXL210E accelerometers are specifically designed to be digital sensors, the advantage of using the analogue signal is its capacity for bandwidth. In fact, the Duty Cycle Modulation (DCM) for the digital signal has a maximum bandwidth of 500Hz, whereas the analogue signal has a bandwidth of up to 6 KHz.
Figure 3.11 shows the analog circuit built in the project [7] on Vero-Board.

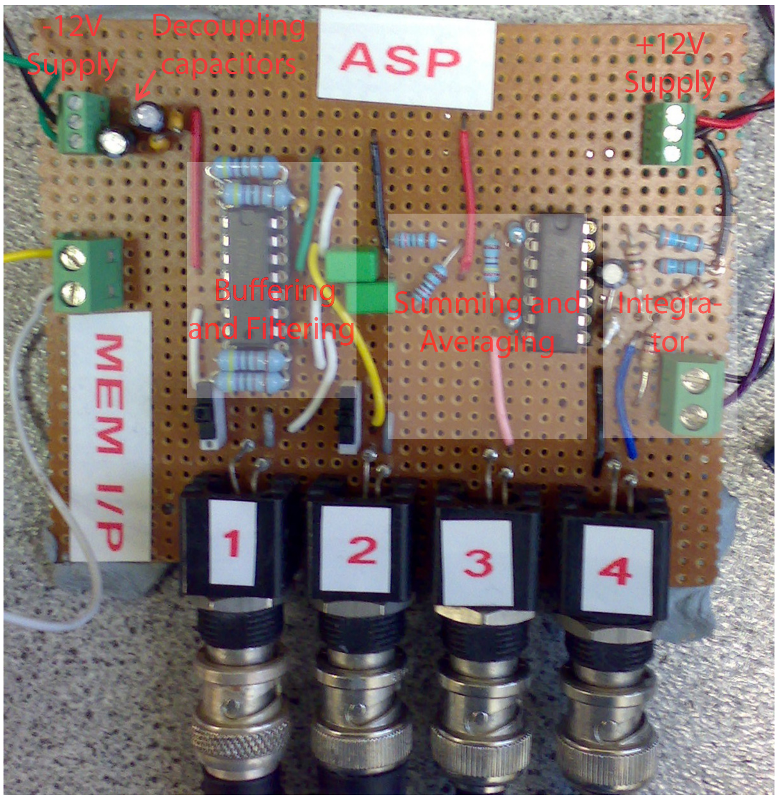


Figure 3.11: Analogue Signal Processing: Vero-Board Circuit.

The coaxial cable connections are labelled 1-4 and these were used to transfer the signals to the dSPACE interface board. Connections 1 and 2 are the individual MEMS buffered signals and by the employment of switches, either the filtered

or unfiltered signal can be connected to the dSPACE board. Connection 3 is the averaged linear acceleration signal and Connection 4 is the differential integrator output. More information on the building of the Vero-Board circuit can be found in [7].
The analogue Vero-Board circuit is shown in Figure 3.12.

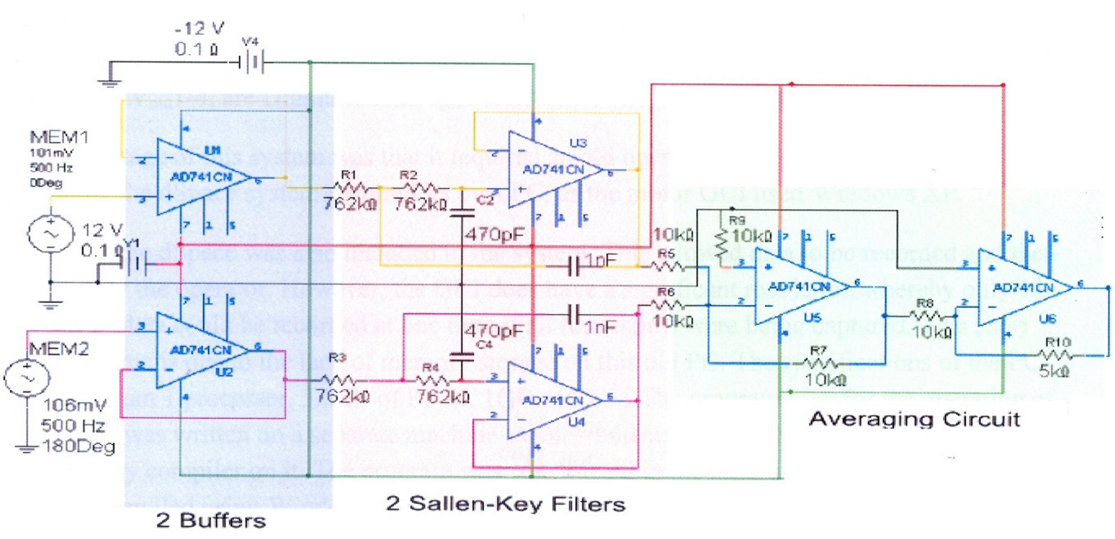


Figure 3.12: Analogue Circuit Diagram. Figure found in [7].

The circuit was implemented to buffer the output signals from the MEMS devices. Decoupling capacitors were integrated into the circuit because this was recommended by the MEMS designers in order to prevent transients from the power supply, as this might disrupt the output of the accelerometers. A second-order low-pass filter *Sallen-Key Filter* was implemented to remove noise introduced from the environment and vibrations, using a cutoff frequency dependent on the resistors. The signals were buffered and filtered individually, and summed. The sum of the signals is used in the averaging circuit, which comprised of an inverting amplifier with a gain of minus one half. In this way, Connection output 3 was an average linear acceleration DC signal with an offset of about 2.5V.

As reported in [7], a basic Resistor-Capacitor (RC) circuit was soldered onto a small piece of Vero-Board and attached to the face of the disk. This circuit was a basic Low-Pass Filter, setting the bandwidth of the MEMS accelerometers, with the additional decoupling capacitors for the supply. Figure 3.13 shows the Low-Pass Filter and Decoupling Capacitors attached on the disk and the position of R_{SET} on the MEMS device. The R_{SET}= 125 kΩ as described in the section 3.3.

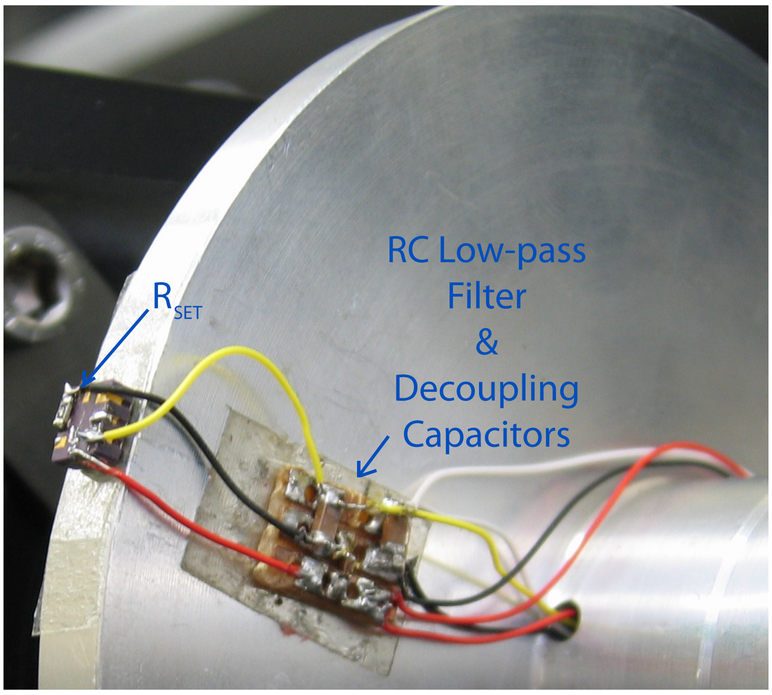


Figure 3.13: Low-Pass Filter and Decoupling Capacitors attached on the disk. Position of R_{SET} on the MEMS device.

3.5 Digital Signal Processing: dSPACE DS1102

This section contains information found in [13].

In the project it was necessary to acquire data from the accelerometers and use them in a digital way, in order to process and store the data acquired. This was possible with the use of Digital Signal Processing (DSP) produced by the dSPACE company. The model of DSP used in the project is DS1102.
The DS1102 is specifically designed for the development of high-speed multivariable digital controllers and real-time simulation. The DS1102 is based on the Texas Instruments TMS320C31 floating-point Digital Signal Processor (DSP) as the main processing unit, providing fast instruction cycle time for numeric intensive algorithms. The DSP has been supplemented by a set of on-board peripherals frequently used in digital control systems. Analog to digital and digital to analog converters, a DSP-microcontroller-based digital-I/O subsystem and incremental sensor interface make the DS1102 an ideal board solution for this project. It contains 128K Words memory, fast enough to allow zero-wait-state operation. Figure 3.14 presents a block diagram of the DS1102.

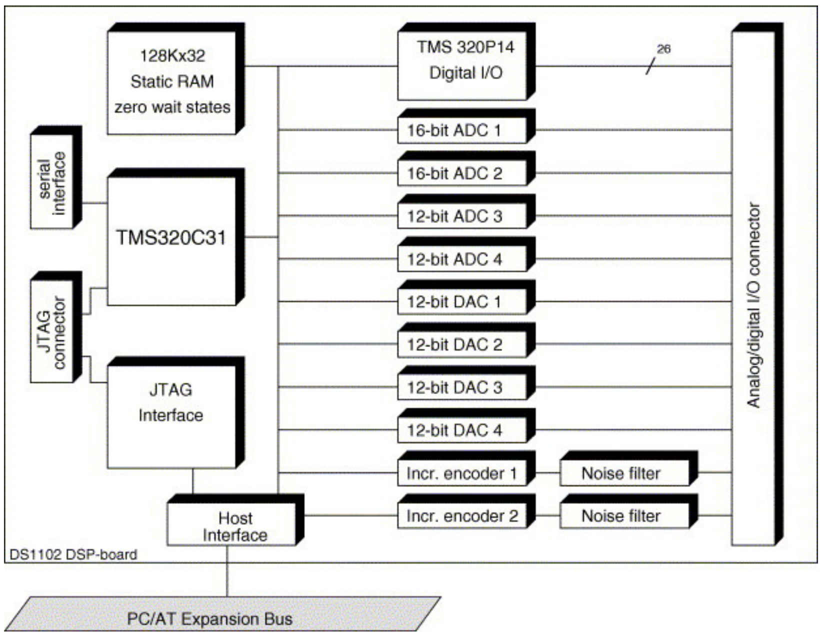


Figure 3.14: Block Diagram of the DS1102. Figure found in [13]

The TMS320C31 supports a total memory space of 16 M 32-bit words including program, data and I/O space. All off-chip memory and I/O can be accessed by the host even while the DSP is running, thus allowing easy system setup and monitoring. The host interface contains a bus-width converter mapping two 16-bit host accesses into a single 32-bit transfer on the DSP-bus to avoid data transfer inconsistencies. The two main parts of the dSPACE system are the interface board and the digital processor, which is directly connected to the computer. Figure 3.15 shows these two main parts. On the board there are two 16-bit Analogue-

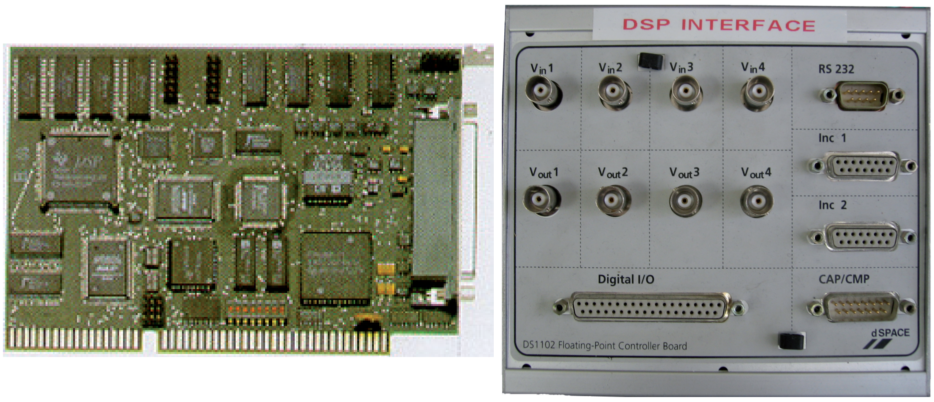


Figure 3.15: DSP dSpace Processor & DSP dSPACE Interface Board.

to-Digital Converters, $V_{in}1$ and $V_{in}2$, and two other inputs, $V_{in}3$ and $V_{in}4$, both 12-bit ADC's. There are 4 Digital to Analogue Converters, connectors $V_{out}1-4$.

The DS1102 dSPACE system requires the use of a computer with a Windows 98 operating system. For this reason the project needs to use two computers, one for the dSPACE system with a Windows 98, GUI, and one for the motor GUI using the Windows XP operating system. The GUI software for dSPACE was included in the project, so that data could be recorded and analysed. Unfortunately there are some restrictions when using this type of computer, as it allows you to store a limited amount of data due to its lack of memory storage. For example, it can register a maximum of 10 seconds data when two signals are read simultaneously. The control program was written in the programming language C and was compiled using a C compiler for the TMS320C31 DSP. The program was subsequently downloaded by DOS to the DSP processor.

The main instructions used in the programs written are:

- *void ds1102_ad_start()*: this function starts all four ADCs contained on the DS1102 and is used in conjunction with the function *ds1102_ad()*;

- *float ds1102_ad(long channel)*: this function returns the ADC input value from the ADC specified by the parameter *channel* which must be within the range 1-4. The ADC data is read subsequently and scaled to a floating-point value in the range -1.0..+1.0. Since the ADC input value is a 16 or 12-bit signed integer, left aligned within a 32-bit data word, the factor 2^{-31} is used for scaling;

- *void ds1102_ad(long channel, float value)*: the contents of *value* is written to the DAC output specified by the parameter *channel*. Valid channel numbers range from 1-4. The output value, which must be within the range -1.0..+1.0, is scaled to a 32-bit integer value by the factor 2^{31} and is written to the respective DAC data register;

- *float ds1102_inc(long channel)*: this function updates the incremental encoder counter output register and returns the position counter value. The parameter *channel* specifies the channel number which must be 1 or 2. The 24-bit position counter value is scaled to a floating point value in the range -1.0..+1.0 by the factor 2^{-31} because the data word is a 24-bit signed integer left aligned within the 32-bit data word;

- *void ds1102_inc_clear_counter(long channel)*: this function resets the selected incremental encoder interface counter specified by the parameter *channel*.

The DS1102 can be interfaced directly by an incremental encoder. Figure 3.16 shows a block diagram of an incremental sensor interface.
The interface contains the lines received for the input signals, a digital noise pulse filter eliminating spikes on the phase lines, a quadrature decoder which

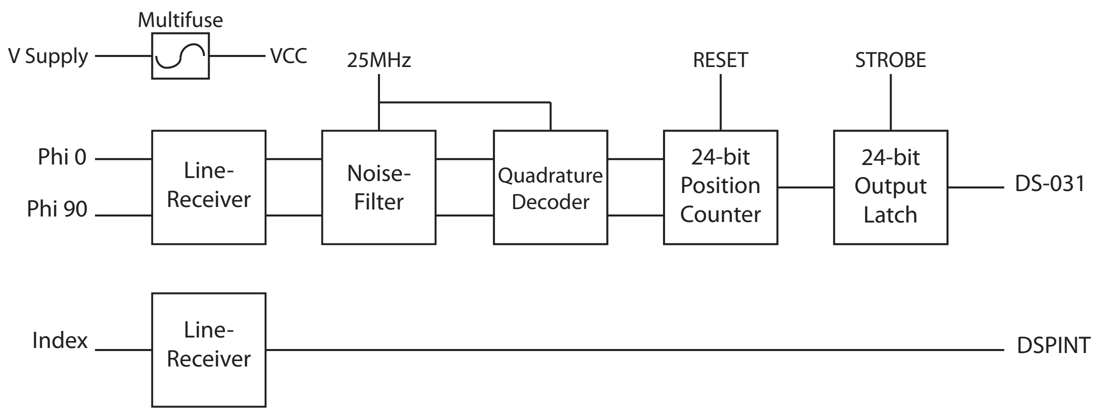


Figure 3.16: Block diagram of an incremental encoder interface.

converts the sensor's phase information to count-up and count-down pulses, a 24-bit counter which holds the current position of the sensor and a 24-bit output latch. Noise pulses shorter than $80ns$ are eliminated by the digital noise pulse filter. Therefore, it was possible to connect the incremental encoder directly to the DS1102 without the use of any other additional circuitry.

3.6 Incremental encoder

An incremental encoder produced by the *Leine & Linde* company has been added to the mechanical system implemented in [7]. Incremental encoders provide a specific number of equally spaced pulses per revolution (PPR) of linear motion. The encoder used in the project provide 4096 PPR and therefore has the following resolution:

$$Resolution\ \delta\vartheta = \frac{2\pi}{4096} = 1.534 \times 10^{-3}\ [rad] \tag{3.10}$$

The encoder is mounted to the motor shaft using an appropriate support attached to the end of the motor's body. Figure 3.17 shows the motor without the encoder and the motor with the encoder mounted.

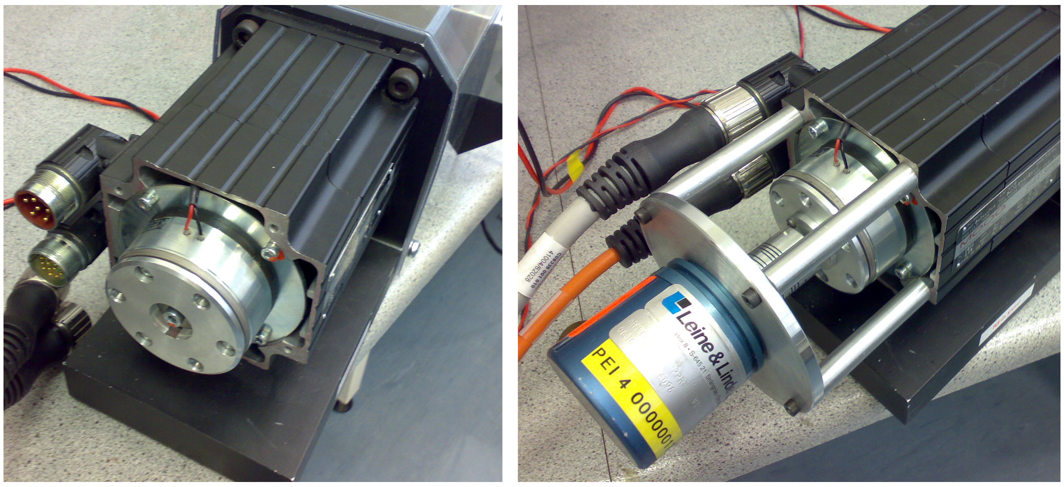


Figure 3.17: Left: Motor without encoder. Right: Motor with encoder and shaft coupling.

It can be seen that the support is needed to fix the encoder to the end of the motor body and a flexible shaft coupling is used to connect the encoder to the motor shaft. The output signals of the encoder are shown in Figure 3.18, and consist of bidirectional signals with the use of a differential line driver.

Unfortunately, one of these signals was not present at the encoder output due to a malfunction, so it was decided to use an inverter chip to obtain the missing signal. The inverter chip used is model MM74HC04N.

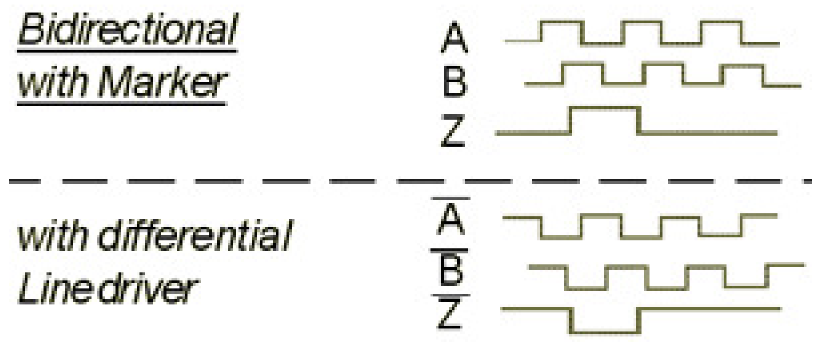


Figure 3.18: Encoder output signal.

As described in the previous section, the encoder is connected directly to the DS1102 board, which also provides the required 5V power to the encoder. To connect the encoder to the interface it was necessary to refer to the manual [13] in order to connect the correct signals to the dSPACE board.

Figure 3.19 shows the mechanical structure of the system. Note the motor where the disc is attached to the MEMS sensor, the encoder with its support, the mercury slip ring and bearing supports to sustain the shaft without friction.

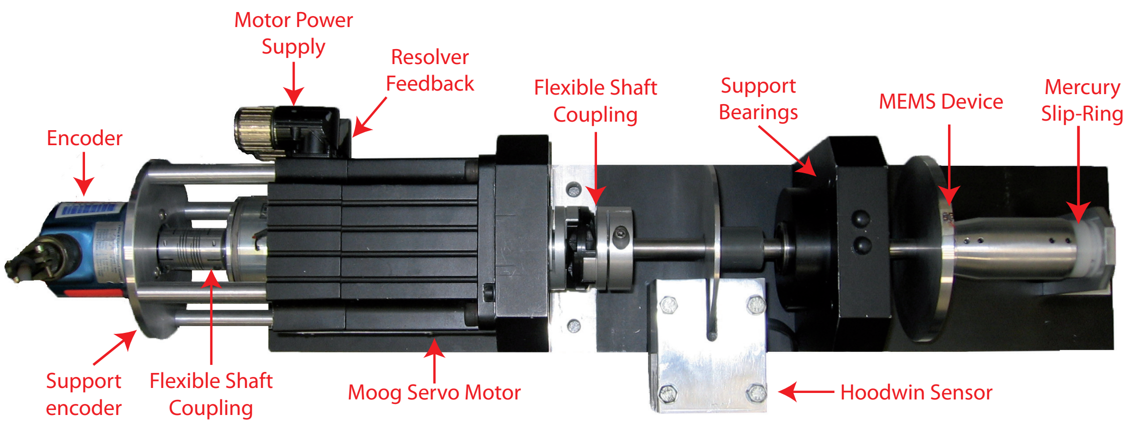


Figure 3.19: Mechanical system features.

Figure 3.20 shows an overview of the system with all its components.

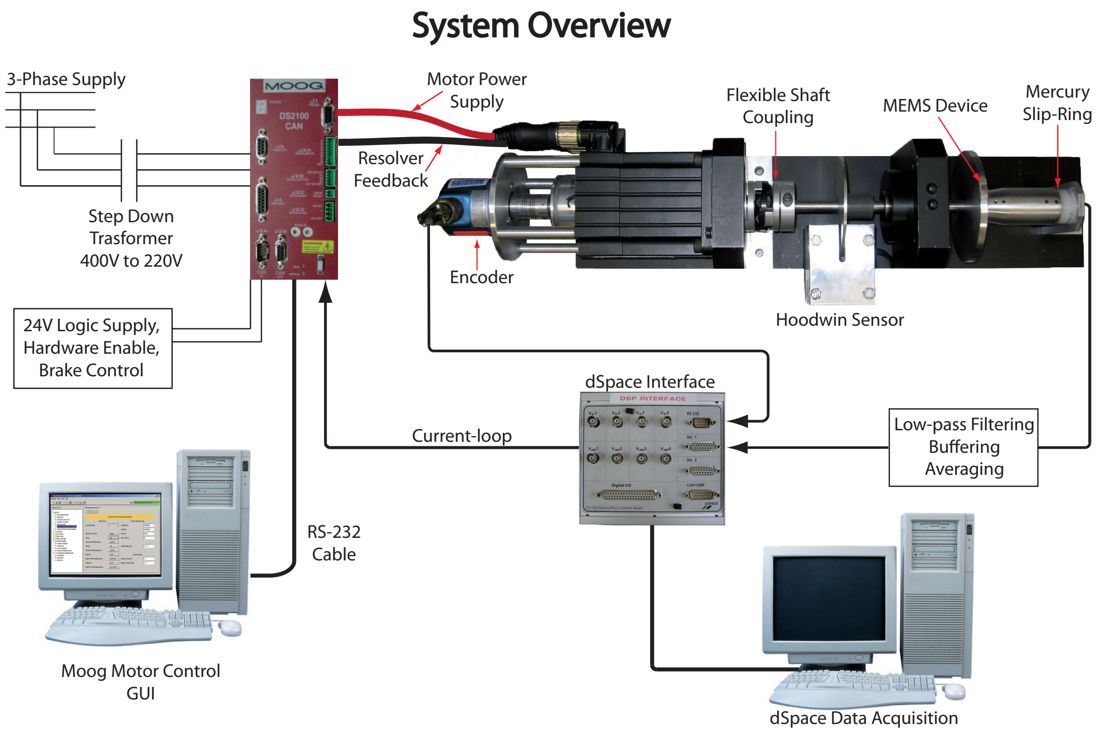


Figure 3.20: Complete system overview.

Chapter 4

Velocity estimation methods

The models in this section are cited in [1].

4.1 Model-based velocity estimation

In most cases, model-based state estimator designs start from the following differential equation:

$$J\ddot{\theta}(t) + b\dot{\theta}(t) = u(t) + d(t) \tag{4.1}$$

where:

- J: nominal value of the inertia;
- b: nominal value of the viscous friction;
- θ: angular position;
- u: input torque;
- d: external disturbance torque.

As suggested in [2], using the estimation of state vector $x = [\theta\ \omega]^T$, and combining a general disturbance observer [14] with a conventional state observer, makes the estimation more robust than the model used in [3] of Kim and Sul.

The system equation is given by

$$\begin{cases} \dot{x}(t) = Ax(t) + B(u(t) + d(t)) \\ y(t) = Cx(t) + q_\theta(t) \end{cases}$$
$$A = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{J} \end{bmatrix}, B = \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix}, C = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{4.2}$$

where $q_\theta(t)$ is due to the quantization error of an encoder.
The quantization error $q_\theta(t)$ is not Gaussian; it is shown to behave as an uncorrelated uniform distribution if the signal is sufficiently complicated and the quantization level is sufficiently small [15].

The states are estimated by the following combined observer:

$$\begin{aligned} \dot{\hat{x}}(t) &= A\hat{x}(t) + B(u(t) + \hat{d}(t)) + F(y(t) - C\hat{x}(t)), \\ \hat{d}(s) &= Q(s)(P_n^{-1}(s)y(s) - u(s)), \end{aligned} \tag{4.3}$$

where $P_n^{-1}(s)$ is the inverse of the nominal transfer function from u to y and $Q(s)$ is a low pass filter with unity gain in order to attenuate the high frequency noise and make $Q(s)P_n^{-1}(s)$ realizable. The state observer gain matrix $F \in \mathbb{R}^{2\times 1}$ can be calculated with Kalman filter equations.

The estimation of the disturbance $\hat{d}(s)$ can be rewritten

$$\hat{d}(s) = \frac{C\psi(s)B}{s + C\psi(s)B} \left(\frac{1 - C\psi(s)F}{C\psi(s)B} y(s) - u(s) \right), \tag{4.4}$$

where $\psi(s) = (sI - A + FC)^{-1}$ and $Q(s)$ and $P_n^{-1}(s)$ in terms of the related parameters in a typical disturbance observer as follows:

$$\begin{aligned} Q(s) &= \frac{C\psi(s)B}{s + C\psi(s)B} = \frac{1}{1 + \sum_{j=1}^{3} k_j s^j}, \\ P_n^{-1}(s) &= \frac{1 - C\psi(s)F}{C\psi(s)B} = Js^2 + bs, \end{aligned} \tag{4.5}$$

where k_j depend on J, b and F. The order of the low-pass filter $Q(s)$ is three and it depends on F, which means that the design of the disturbance observer is coupled with the design of the state estimator. For discrete-time implementation, the system equation 4.2 can be rewritten as

$$\begin{cases} x(k+1) = A_m x(k) + B_m(u(k) + w(k)) \\ y(k) = Cx(k) + q_\theta(k) \end{cases} \tag{4.6}$$

where the subscript m stands for the model-based scheme, w is a perturbation term and A_m and B_m are zero-order-hold discrete time equivalents of A and B, i.e.

$$A_m = \begin{bmatrix} 1 & \frac{J}{b}\left(1 - e^{-\frac{b}{J}T_s}\right) \\ 0 & e^{-\frac{b}{J}T_s} \end{bmatrix}, B_m = \frac{1}{b}\begin{bmatrix} T_s - \frac{J}{b}\left(1 - e^{-\frac{b}{J}T_s}\right) \\ 1 - e^{-\frac{b}{J}T_s} \end{bmatrix} \tag{4.7}$$

For model (4.6), the state observer for model-based velocity estimation is given by

$$\hat{x}_m(k+1) = A_{mc}\hat{x}_m(k) + B_{mc}(u(k) + \hat{w}(k)) + F_m y(k+1), \tag{4.8}$$

where

$$\begin{aligned} A_{mc} &= (I - F_m C)A_m, \\ B_{mc} &= (I - F_m C)B_m, \end{aligned} \tag{4.9}$$

and F_m is obtained by the Kalman filter equation. In fact, it is calculated by a discrete time algebraic Riccati equation as follows:

$$F_m = MC^T \left[CMC^T + V\right]^{-1}, \tag{4.10}$$

$$M = A_m M A_m^T + B_m W B_m^T - A_m M C^T \left[CMC^T + V\right]^{-1} CMA_m^T, \tag{4.11}$$

where W and V are the variances of $\tilde{w}(k) = w(k) - \hat{w}(k)$ and $q_\theta(k)$, respectively, and M is the one-step-ahead prediction error covariance matrix.

From (4.4) and (4.8) the error dynamics equations can be written with the following equation:

$$\tilde{x}_m(k+1) = A_{mc}\tilde{x}_m(k) + B_{mc}\tilde{w}(k) - F_m q_\theta(k+1) \tag{4.12}$$

Observations on the dynamic error are made in the next section with the use of data.

4.2 Kinematic Kalman filter (KKF)

The Kalman filter based on the kinematic model is called the kinematic Kalman filter (KKF) [5]. The kinematic model relates angular acceleration $\alpha(t)$ to position θ by

$$\ddot{\theta}(t) = \alpha(t). \tag{4.13}$$

Considering the real angular acceleration $\alpha(t)$ as the sum of the measurement $a(t)$ and its noise component $w_a(t)$, a state space representation of the kinematic model has acceleration as an input and the encoder measurement as the system output. Since the encoder measurements are obtained only intermittently, it is best to describe the kinematic model in the discrete-time domain.

The zero-order-hold equivalent of (4.13) is

$$\begin{gathered}\begin{cases} x(k+1) = A_k x(k) + B_k(a(k) + w_a(k)) \\ y(k) = Cx(k) + q_\theta(k) \end{cases} \\ A_k = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix}, B_k = \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \end{bmatrix}. \end{gathered} \tag{4.14}$$

where the subscript k stands for the kinematic model and $w_a(t)$ is the noise of the accelerometer. The accelerometer noise $w_a(t)$ is correctly modelled as a zero-mean Gaussian white noise by nature.

The state estimator is given by

$$\hat{x}_k(k+1) = A_{kc}\hat{x}_k(k) + B_{kc}a(k) + F_k y(k+1), \tag{4.15}$$

where

$$\begin{aligned} A_{kc} &= (I - F_k C)A_k; \\ B_{kc} &= (I - F_k C)B_k. \end{aligned} \tag{4.16}$$

The optimal estimator gain F_k is calculated by a discrete time algebraic Riccati equation as follows:

$$F_k = MC^T \left[CMC^T + V\right]^{-1}, \tag{4.17}$$

$$M = A_k M A_k^T + B_k W_a B_k^T - A_k M C^T \left[CMC^T + V\right]^{-1} \times CMA_k^T. \tag{4.18}$$

Note that the variance W_a of the input noise $w_a(k)$ is readily obtained from the sensor specification while the perturbation term $\hat{w}(k)$ is a signal to be estimated.

From (4.14) and (4.15) the error dynamics equations can be written with the following equation:

$$\tilde{x}_k(k+1) = A_{kc}\tilde{x}_k(k) + B_{kc}w_a(k) - F_k q_\theta(k+1) \tag{4.19}$$

Observations on the dynamic error are made in the next section with the use of data.

Chapter 5

Design and Simulation

As a first step before of the design of controllers and estimators, it is necessary to determine an approximation of the parameters that are not included in the project, such as the coefficient of friction and the coefficient of viscous damping. The second step, with the knowledge of these coefficients, is to design the controller and estimators.

For this type of motor, the job of designing the controller would be much more complex without using the Moog DS2100 servosystem, since it would be necessary to design a controller for the PWM inverter, as in [3]. Using the current loop connected to the DS2100 has made it possible to consider the torque to be proportional to current supplied, thus simplifying the design of the controller.

5.1 Estimated Friction and Viscosity Damping

Knowledge of friction and viscosity present in the system are an important part of the control system and estimator design. However, they are also extremely difficult to determine, because they have some non-linear components and are difficult to measure in the system. A simulation was carried out on a motor system in order to investigate the behaviour of this friction model.

As described in [16], friction exists to some extent in all mechanical systems where surfaces are in contact and free to move. Friction is usually analysed as two components: linear and nonlinear friction. Non linear friction is often treated as having two distinct components: static friction (stiction) element which is the level of torque required to cause breakaway, and a coulomb friction torque which exists between surfaces during relative motion. A common model of nonlinear friction, used by Tung et al [17], postulates a velocity-dependent exponential decay of the level of the friction from its static value to a constant level of coulomb friction at high velocity.

$$T_{fric} = T_c \operatorname{sgn}(\omega) + T_{\exp} e^{-\beta|\omega|} \operatorname{sgn}(\omega) \tag{5.1}$$

Friction is illustrated in Figure 5.1.

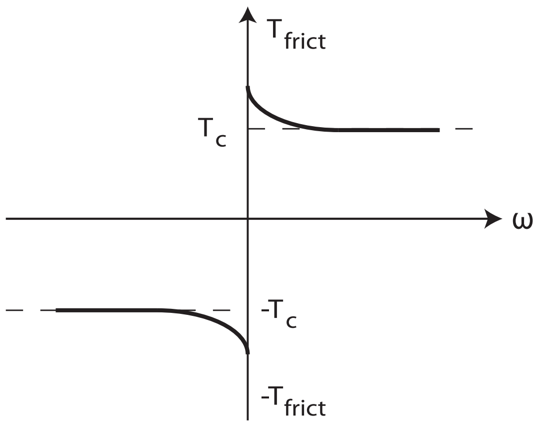


Figure 5.1: Relationship of friction and the velocity. Figure found in [16].

The sum of the coefficients, T_c and T_{exp}, represents the static friction level, T_{stic} and β is the slip constant which governs the region between stiction and coulomb friction. This friction characteristic is common to mechanical systems in which a thin lubrication layer exists on the bearing [18].

To estimate these parameters, the motor was controlled by a sine wave signal with amplitude of 50mV and frequency of 200mHz. Figure 5.2 left, shows the voltage and the corresponding current, the input to the system.

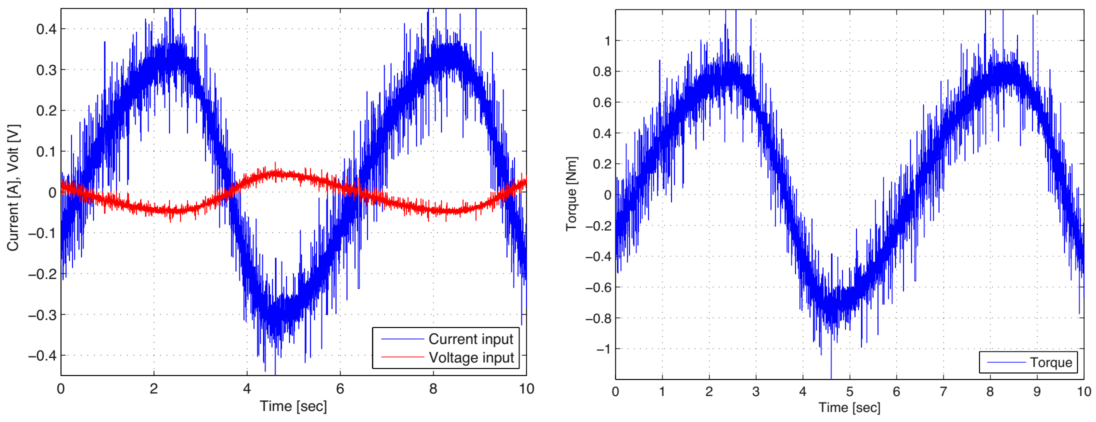


Figure 5.2: Left: Sine wave voltage input and corresponding current input. Right: Sine wave torque input.

This important correspondence is described in Section 3.2 and some tests conducted using the GUI and the current-loop.
The relationship is given by the following equations:

$$\begin{aligned} volt_{input} &= volt_{ref} - 2.425; \\ current_{input} &= -\frac{volt_{input} \cdot 17}{2.425}. \end{aligned} \tag{5.2}$$

In the equation (5.2), $current_{input}$ is the input signal to the system, while $volt_{ref}$ is the input signal present in the current-loop to the servosystem. Then $volt_{ref}$ must be converted using the relationship of Section 3.2. Therefore, when a positive value of $volt_{input}$ is obtained, the output of the converter, leads to a negative current, using the relationship in (5.2).

Knowing the current, it is possible to calculate the torque applied to the motor using the following equation for permanent magnet synchronous motors:

$$m = \frac{3}{2} p \Lambda_{mg} i_q \tag{5.3}$$

where Λ_{mg} represents the maximum flux in each phase due to the permanent magnet and p represents the number of motor poles. The value of $p\Lambda_{mg}$ was obtained by performing several tests on the system, comparing the current and torque values read by the GUI. The value is:

$$p\Lambda_{mg} = 1.58 \tag{5.4}$$

At this point, it was possible to calculate the corresponding sine wave torque that excites the system. The torque is shown in Figure 5.2 right. The friction torque

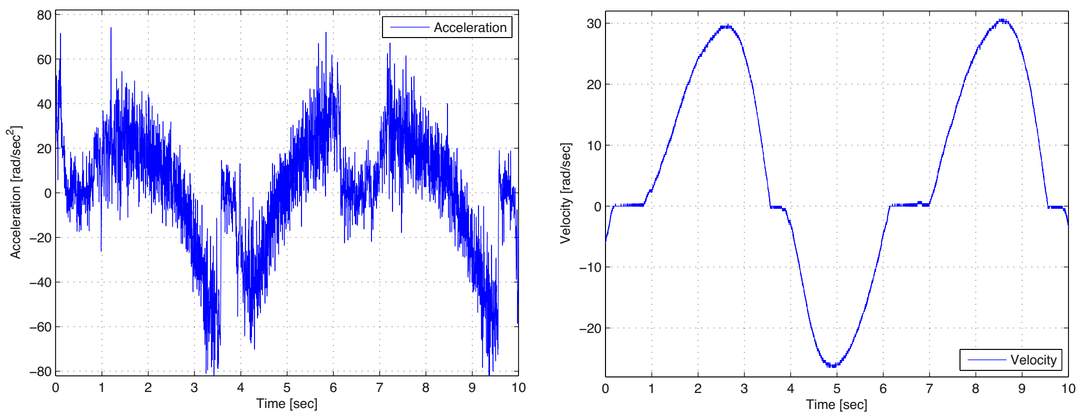


Figure 5.3: Left: Acceleration measured. Right: Velocity measured.

follows the input torque until the input reaches the stiction level indicating that no torque excites the system, and so the system remains at rest, as shown in Figure 5.3 right. Motion commences when the input torque exceeds the stiction level and the friction torque then falls to the coulomb level. In this way, it was possible to estimate the value of the torque load; the result is $m_L = 0.3$ [Nm]. When the acceleration and velocity measured by the system are known, Figure 5.3, the torque relationship of the mechanical load can be used:

$$m = m_L + B\omega_m + J\frac{d\omega_m}{dt} \tag{5.5}$$

In this way, it was possible to obtain an approximation of the viscous damping (parameter B), which amounted to about B = 0.015 [Nm(s/rad)].

This is only a simple approximation of viscous damping, because there are many simplifications and measurement errors in the model. There are other much more efficient and effective methods of finding this parameter, but they are still approximations. One of these methods is to measure the damping coefficient using constant speed experiments which involve running the motor at constant speed in response to numerous constant torque demands. At constant speed, acceleration torque is eliminated from the dynamic equation and then relationship can be derived for the friction terms. Therefore, finding a realistic value of friction parameters to reality is extremely difficult. In fact multi-value friction and viscous damping characteristics that depend on surface materials and lubrication are sometimes evident [19].

Typically, the viscosity is very small and is often overlooked. In this case, the coefficient found has a non trivial magnitude. This may be due to the fact that there are a large number of moving parts in the system, as can be seen in Figure 3.19.

5.2 Designing a PI speed controller

Firstly, because it is designed for speed control for an isotropic motor it was necessary to design two different controllers, one for speed and one for torque control. The procedures for the design of these controllers are referred to in [20].This section shows the main parts for their design. For more information, see [20].

Figure 5.4 shows a block diagram of the speed control of a brushless motor. For convenience, the diagram shows unity gain feedback, which is marked with

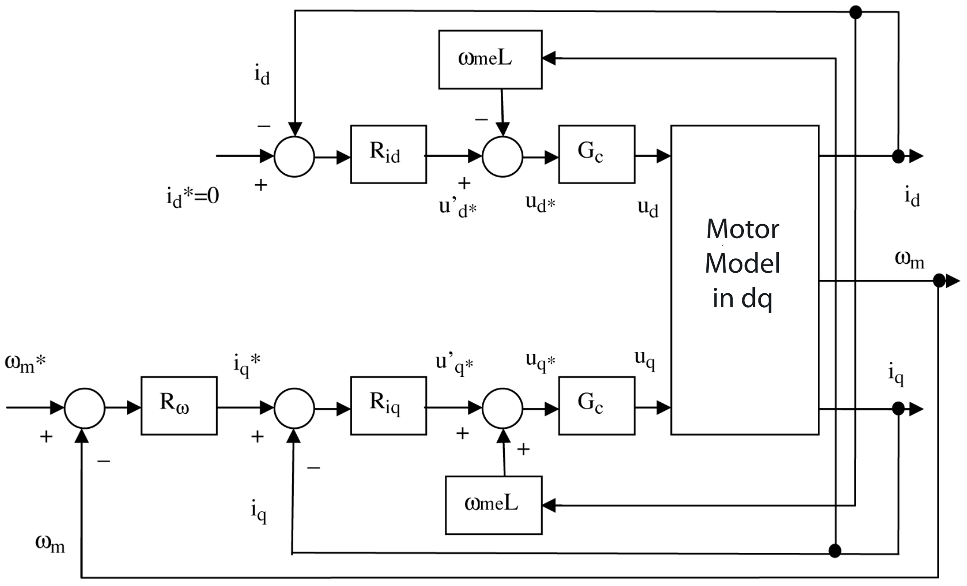


Figure 5.4: Block Diagram of the control of a isotropic brushless motor. Figure found in [20].

. The speed reference ω_m is compared with the measured speed and through the speed controller R_ω it produces the quadrature current reference i_q* which is proportional to torque. The direct current reference i_d* is maintained at zero. The current i_d does not contribute to motor torque and running at speeds above base speed is not of interest for the brushless isotropic rotor. The two current references are compared with their measurements and the errors are drawn from the current regulators to produce the voltage references u_q* and u_d*. In practice, the two voltage references are converted into the corresponding voltage references

u_α* and u_β* which, through PWM control of the inverter, apply voltages to the power the motor. Similarly, the phase currents are measured and converted into the components i_d and i_q.

From the point of view of control design it is possible that the reference voltage u_d* and u_q* produce, with the dynamics that characterize the inverter, and similar voltages u_d and u_q applied to the motor which yields the resulting currents i_d and i_q. With these assumptions, the diagram in Figure 5.4 includes $G_c(s)$, which is the transfer function of the inverter:

$$G_c(s) = \frac{U(s)}{U^*(s)} = \frac{1}{1 + s\tau_c} \tag{5.6}$$

with τ_c being linked to the period T_c of the PWM modulation. The motor is described by a single block that contains the model. The design of current controllers is complicated by the fact that the two loops are not mutually independent, but influence each other due to this cross-coupling between the axes of the motor (d and q). If the time constant τ_c of the inverter is small, as is usual, compared to other time constants in the system, it is possible to eliminate the mutual coupling between the d and q axes (decoupling). Figure 5.5 shows the block diagram of current control after decoupling.

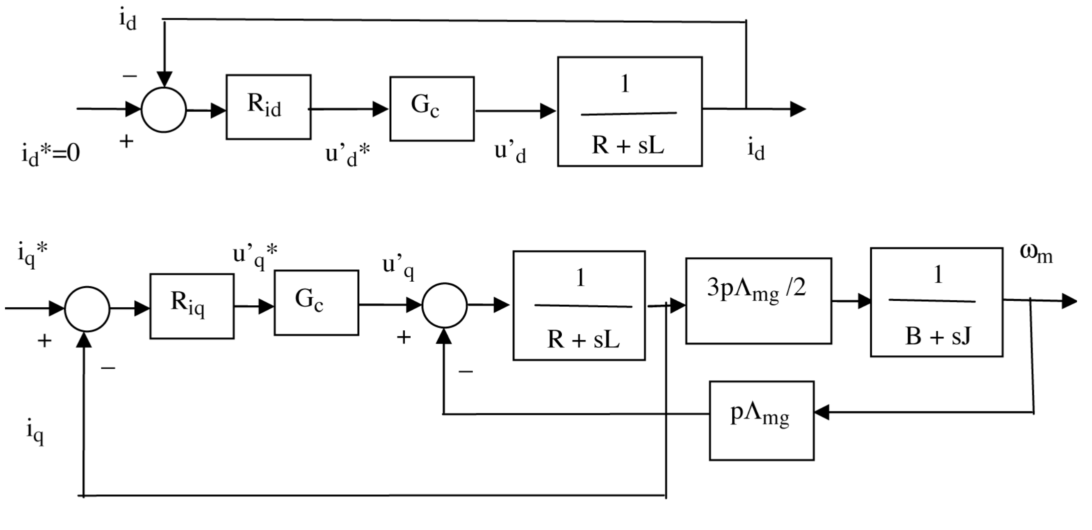


Figure 5.5: Block diagram of current control after decoupling. Figure found in [20].

The control for the brushless motor thus obtained is similar to a DC motor drive, with the d phase which takes on the role of the field circuit and phase q of the armature. In other words, the design of the PI controller is greatly simplified.

For the design of controllers and for the subsequent design of the estimators, the data of the overall system are presented in Tables 5.1.

Type	Symbol	Value	Units
Nominal Torque	M_o	2.6	Nm
Nominal speed	n_N	5500	rpm
Maximum speed	n_{max}	8000	rpm
Nominal current	I_o	4.8	Arms
Peak current	I_p	15	Arms
Output power	P_N	0.95	kW
Torque constant	k_t	0.56	Nm/Arms
Voltage constant	k_e	34.2	Vrms/krpm
Rotor inertia with resolver	J	2.09	kg cm^2
Viscous damping	b	0.015	Nm(s/rad)
Winding resistance at 25°C	R	2.6	Ohm
Winding inductance (phase to phase)	L	5.8	mH
Motor pole count	n_p	12	
Striction level	F_s	0.3	Nm
Noise variance	W_a	5	$(rad/s^2)^2$
Encoder counter	N	2^{12}	$(ppr)^a$
Sampling time	T_s	0.001	sec

Table 5.1: Experimental conditions.

5.2.1 Design of current control

As a first step in the design of current control, assume the following constants dependent on motor parameters and drive. The values can be taken from Table 3.1 or 5.1.

- $\tau_a = \frac{L}{R} = 0.0021$, electrical time constant;
- $\tau_m = \frac{J}{B} = 0.0139$, mechanical time constant;
- $\tau_{m1} = \frac{JR}{K_e^2} = 0.0018$, electromechanical time constant;
- $\tau_c = 0.001$, time constant of the inverter.

Figure 5.6 shows a diagram of current control in the s domain, where $R_i(s)$ is the PI controller and $Y(s)$ is the transfer function of the model that links the current to voltage.

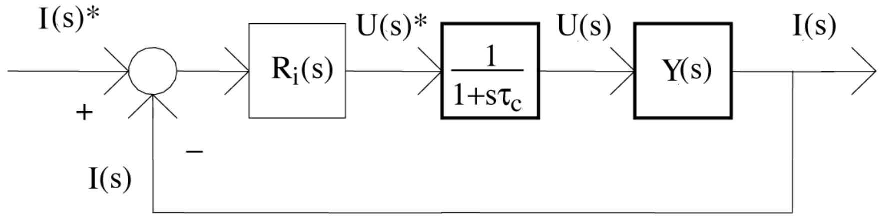


Figure 5.6: Current control scheme in the domain of s. Figure found in [20].

The following open-loop transfer function (without the regulator) is considered:

$$GH_R(s) = \frac{1}{1+s\tau_c}Y(s) = \frac{B}{K_e^2}\frac{(1+s\tau_m)}{(1+s\tau_c)(1+s\tau_a)(1+s\tau_{m1})} \tag{5.7}$$

The steady state characteristics of $GH_R(s)$ are obtained by examining the magnitude and phase function $GH_R(j\nu)$ in the variable ν obtained by substituting $s = j\nu$ in (5.7).

The current PI controller is defined by the relation:

$$R_i(s) = K_{Pi} + \frac{K_{Ii}}{s} = K_{Pi}\frac{1+s\tau_{Ri}}{s\tau_{Ri}} = K_{Ii}\frac{1+s\tau_{Ri}}{s}, \tag{5.8}$$

where $\tau_{Ri} = K_{Pi}/K_{Ii}$ is the current regulator time constant. Therefore, the open-loop transfer function $GH(s)$ is obtained by multiplying the $GH_R(s)$ by $R_i(s)$.

After defining the value of the crossover frequency ν_i and the time constant τ_{Ri} of the current regulator, the value of K_{Pi} is provided by the following equation:

$$1 = K_{Pi}\frac{B}{K_e^2}\frac{\sqrt{(1+(\nu_i\tau_m)^2)}\sqrt{(1+(\nu_i\tau_{Ri})^2)}}{\nu_i\tau_{Ri}\sqrt{(1+(\nu_i\tau_c)^2)}\sqrt{(1+(\nu_i\tau_a)^2)}\sqrt{(1+(\nu_i\tau_{m1})^2)}} \tag{5.9}$$

Therefore, the phase margin m_ϕ, which does not depend on K_{Pi} but on the crossover frequency, is calculated as

$$m_\phi = \arg\left[GH(j\nu_i)\right] + \pi \tag{5.10}$$

that is:

$$m_\phi = \arctan(\nu_i\tau_{Ri}) + \arctan(\nu_i\tau_m) - \arctan(\nu_i\tau_c) - \arctan(\nu_i\tau_a) - \arctan(\nu_i\tau_{m1}) + \frac{\pi}{2} \tag{5.11}$$

The values in Table 3.1 produced:

$$GH_R(s) = \frac{172413\ (\mathrm{s} + 71.77)}{(\mathrm{s} + 1000)(\mathrm{s} + 555.7)(\mathrm{s} + 465.5)} \tag{5.12}$$

By setting a crossover frequency as $\nu_i = 1000$, and the time constant τ_{Ri} approximately equal to the electrical time constant, the following controller was obtained:

$$R_i(s) = \frac{9.36(s + 465.5)}{s} \tag{5.13}$$

Therefore, the following open-loop transfer function was obtained:

$$GH(s) = \frac{1613773\ (\mathrm{s} + 465.5)\ (\mathrm{s} + 71.77)}{\mathrm{s}\ (\mathrm{s} + 1000)\ (\mathrm{s} + 555.7)\ (\mathrm{s} + 465.5)} \tag{5.14}$$

which has a phase margin $m_\phi = 1.2210[rad] = 69.9573°$.

Figure 5.7 shows the frequency response of the transfer function $GH(s)$.

In this way, the proportional gain and integral gain of the controller were calculated from the equations 5.9 and 5.13, giving:

$$\begin{aligned} K_{Pi} &= 9.3599; \\ K_{Ii} &= \frac{K_{Pi}}{\tau_{Ri}} = 4357. \end{aligned} \tag{5.15}$$

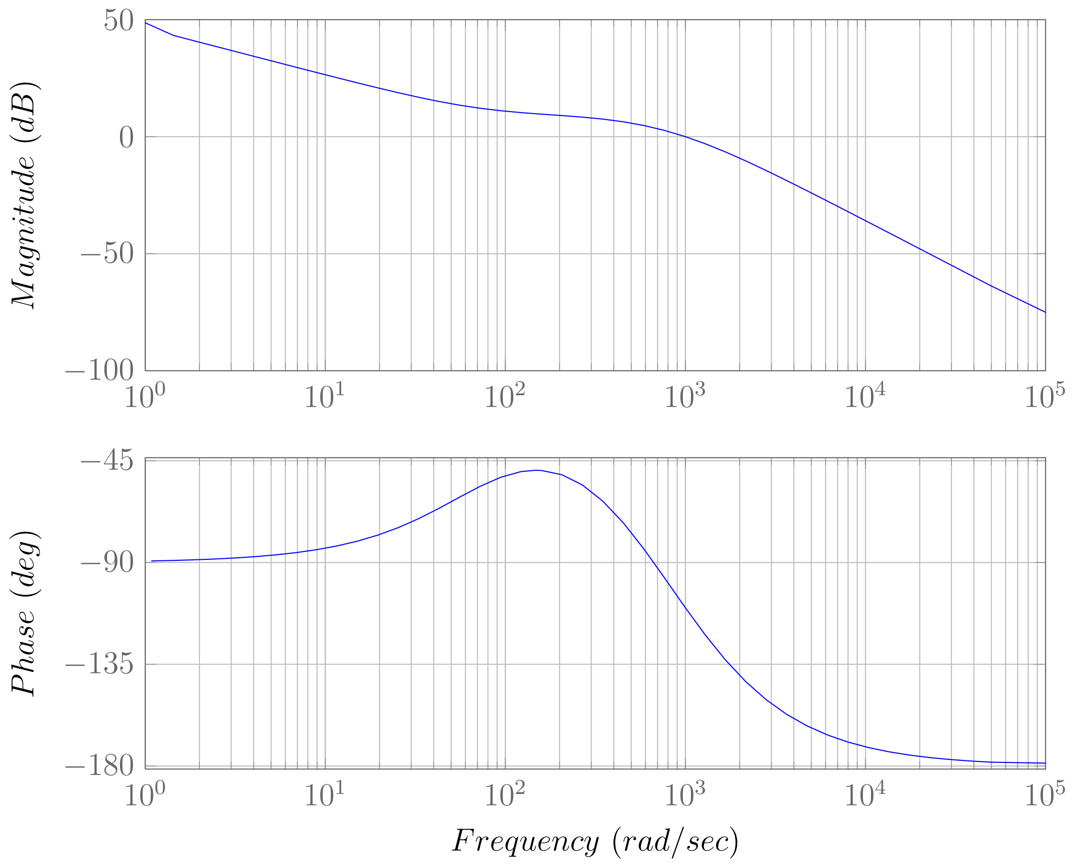


Figure 5.7: Bode diagram of the open-loop transfer function of the current controller.

5.2.2 Design of speed control

The next step was speed control design. The speed control loop, with the addition of an inertial load m_L, is shown in Figure 5.8, where $R_\omega(s)$ is the speed controller and $W_i(s)$ is the transfer function of the closed loop current control.

A (Bode) approximation that is often used for the frequency response of $W_i(j\nu)$ is as follows:

$$W_i(j\nu) = \frac{G(j\nu)}{1+G(j\nu)H} = \begin{cases} \frac{1}{H} & if \quad \nu < \nu_i \\ G(j\nu) & if \quad \nu > \nu_i \end{cases} \tag{5.16}$$

where it is assumed for convenience that H (static gain feedback) does not depend on s.

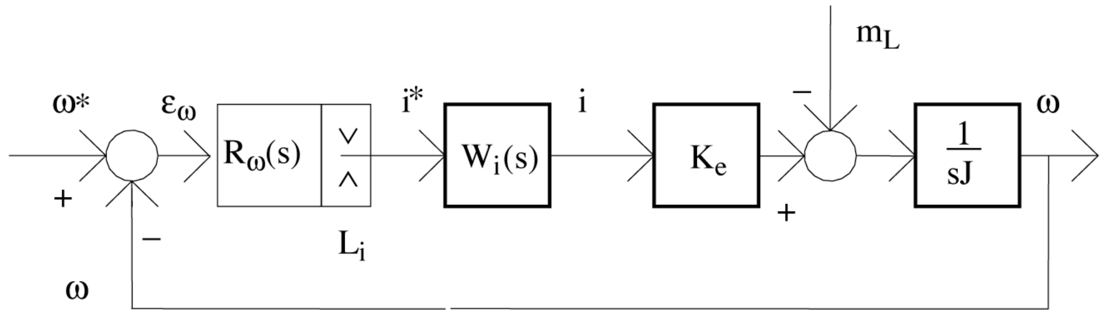


Figure 5.8: Block Diagram of Speed Control. Figure found in [20].

Therefore, the transfer function is approximated as:

$$W_i(s) = \frac{1}{\left(1 + \frac{s}{\nu_i}\right)(1 + s\tau_c)} \tag{5.17}$$

The PI speed controller is characterized by the $K_{P\omega}$ gain and a time constant $\tau_{R\omega}$. The controller has the following transfer function:

$$R_\omega(s) = K_{P\omega} + \frac{K_{I\omega}}{s} = K_{P\omega}\frac{1 + s\tau_{R\omega}}{s\tau_{R\omega}} \tag{5.18}$$

The open-loop function $GH(s)$ is as follows:

$$GH(s) = \frac{K_{P\omega}K_e}{\tau_{R\omega}J}\frac{(1 + s\tau_{R\omega})}{s^2\left(1 + \frac{s}{\nu_i}\right)(1 + s\tau_c)} \tag{5.19}$$

Therefore, the phase margin m_φ is calculated by the following equation:

$$m_\varphi = \arctan(\nu_\omega\tau_{R\omega}) - \arctan(\nu_\omega\tau_c) - \arctan(\nu_\omega\tau_a) - \arctan(\nu_\omega\frac{1}{\nu_i}) \tag{5.20}$$

A practical way for determining the controller is to choose ν_ω, i.e. the crossover frequency of the speed controller as the geometric mean between $1/\tau_{R\omega}$ and ν_i, and set these two at a distance of about one decade (method symmetric optimal). A different way is to impose the crossover frequency ν_ω equal to about half the crossover frequency ν_i and $1/\tau_{R\omega}$ at least a decade less than this. After some simulations using MATLAB, the following values were chosen:

- $\tau_{R\omega} = 0.04$, time constant of speed controller;
- $\nu_\omega = 30$, crossover frequency of the speed controller.

These values give a speed control:

$$R_\omega = \frac{8.6 \times 10^{-3}(s+25)}{s} \tag{5.21}$$

and the following transfer function:

$$GH(s) = \frac{23.06 \times 10^{6}(s+25)}{s^2(s+1000)^2} \tag{5.22}$$

which has a phase margin $m_\phi = 0.81[rad] = 46.75°$.

Figure 5.9 shows the frequency response of the transfer function $GH(s)$.

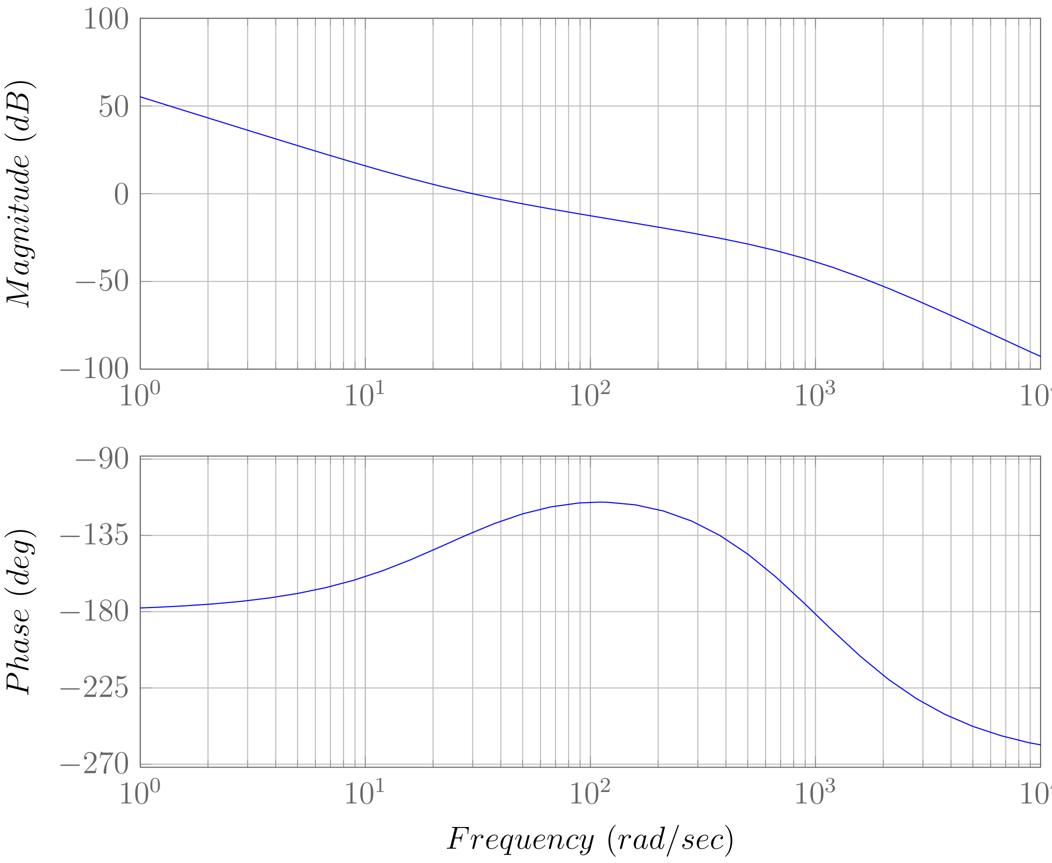


Figure 5.9: Bode diagram of the open-loop transfer function of the speed controller.

The controller parameters were derived from 5.19 and were:

$$\begin{aligned} K_{P\omega} &= 0.0086; \\ K_{I\omega} &= \frac{K_{P\omega}}{\tau_{R\omega}} = 0.2152. \end{aligned} \tag{5.23}$$

To test the response of the controller design, the system was simulated using the *Simulink* block diagram, shown in Figure 5.10. Figure 5.11 shows in detail the PI current control, which was inserted into the "subsystem" in Figure 5.10.

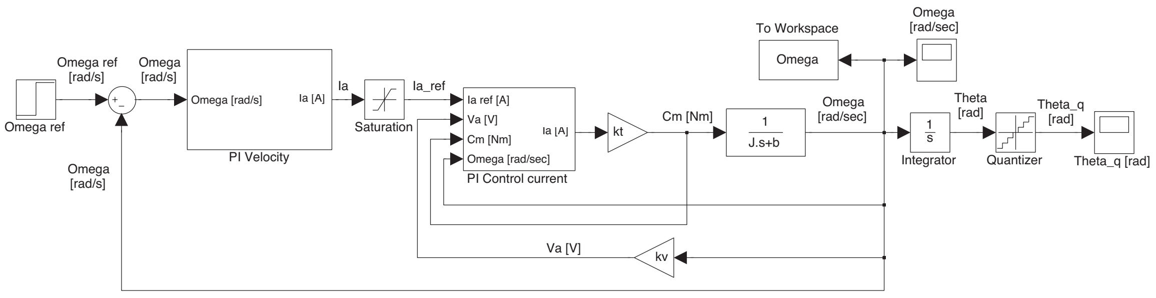


Figure 5.10: Simulink block diagram of the speed control.

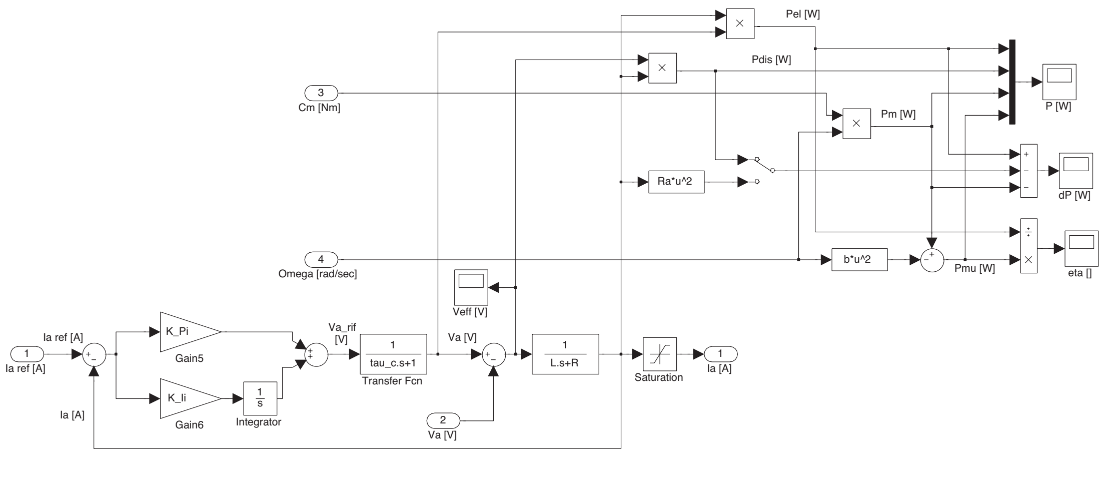


Figure 5.11: Simulink block diagram of current control.

Figure 6.28 shows the simulated response to a speed step input of amplitude 10 [rad/sec]. The simulated response has a rise time equal to 0.3 [sec], the time required for a signal to change from 10% to 90% of the step height. The response did not exhibit overshoot and was stabilized to the reference signal without oscillation.

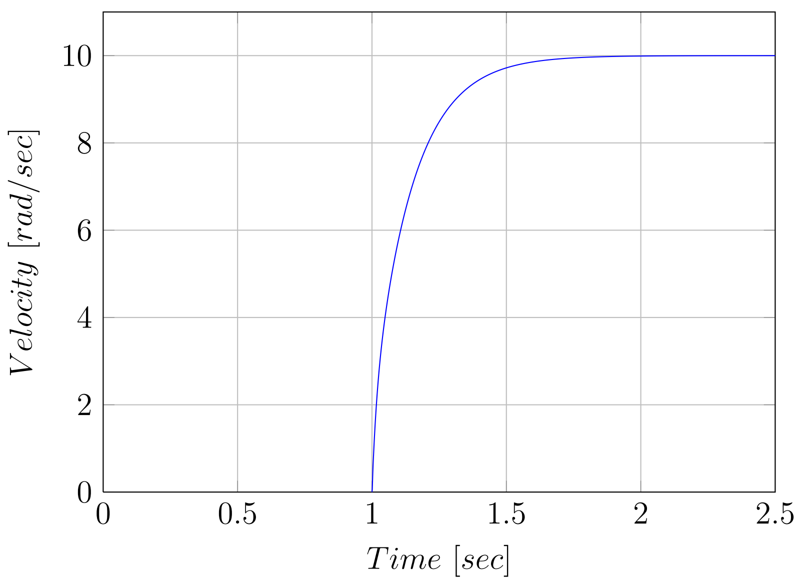


Figure 5.12: System response at a step speed of amplitude 10 [rad/sec].

The controller could be made even faster by changing the parameters of the controllers, but this was not the main part of the project and special design specifications were not required.

5.3 Design Model-Based Estimator and Kalman Filter

Firstly, the Model-Based estimator has been designed using the formulae described in Section 4.1 and the values reported in Table 5.1.

The matrices of the Model-Based estimator in (4.7) are calculated as

$$\begin{aligned} A_m &= \begin{bmatrix} 1 & \frac{J}{b}\left(1 - e^{-\frac{b}{J}T_s}\right) \\ 0 & e^{-\frac{b}{J}T_s} \end{bmatrix} = \begin{bmatrix} 1 & 0.001 \\ 0 & 0.9307 \end{bmatrix}, \\ B_m &= \frac{1}{b}\begin{bmatrix} T_s - \frac{J}{b}\left(1 - e^{-\frac{b}{J}T_s}\right) \\ 1 - e^{-\frac{b}{J}T_s} \end{bmatrix} = \begin{bmatrix} 0.0023 \\ 4.617 \end{bmatrix}. \end{aligned} \tag{5.24}$$

where A_m and B_m are zero-order-hold discrete time equivalents of A and B in (4.2). As is shown in [1], q_θ is bounded by the encoder resolution $\delta\theta$ in (3.10). This gives the following approximate output noise variance:

$$V = \frac{q_\theta^2}{12} = \frac{\delta\theta^2}{12} = 1.9609 \times 10^{-7}\ [rad^2] \tag{5.25}$$

Assuming the disturbance observer cannot catch the stiction nor the cogging torque, the variance of W of the disturbance estimation error $\tilde{w}(k)$ was selected using the stiction level F_s, i.e:

$$W = \frac{F_s^2}{12} = 0.0075\ [(\text{Nm})^2] \tag{5.26}$$

In practice, a suitable choice for W will be determined by further trial and error. Using these values, it was possible to calculate the following Q matrix:

$$Q = B_m W B_m^T = \begin{bmatrix} 4.0931 \times 10^{-8} & 8.0895 \times 10^{-5} \\ 8.0895 \times 10^{-5} & 0.1599 \end{bmatrix} \tag{5.27}$$

Therefore, it was possible to solve the discrete-time algebraic Riccati equation in (4.18), using the MATLAB instruction:

$$[M, s, e] = dare(A_m', C', Q, V);$$

giving the following one-step prediction error covariance matrix:

$$\mathrm{M} = \begin{bmatrix} 5.0414 \times 10^{-7} & 3.1056 \times 10^{-4} \\ 3.1056 \times 10^{-4} & 0.3033 \end{bmatrix}. \tag{5.28}$$

The observer gain was calculated using equation (4.10):

$$F_m = MC^T \left[CMC^T + V\right]^{-1} = \begin{bmatrix} 0.72 \\ 443.5102 \end{bmatrix}. \tag{5.29}$$

Therefore, using equation (4.9) and observer gain, calculated:

$$\begin{aligned} A_{mc} &= (I - F_m C) A_m = \begin{bmatrix} 0.28 & 2.7023 \times 10^{-4} \\ \text{- } 443.5102 & 0.5028 \end{bmatrix}, \\ B_{mc} &= (I - F_m C) B_m = \begin{bmatrix} 6.5421 \times 10^{-4} \\ 3.5809 \end{bmatrix}. \end{aligned} \tag{5.30}$$

Regarding the disturbance observer in equation (4.4), disturbance $\hat{d}$ can be represented by u and y as:

$$\hat{d}\left(s\right) = D\left(s\right)\left(Q\left(s\right)y\left(s\right) - u\left(s\right)\right) \tag{5.31}$$

In other words, $Q(s)$ is nothing but a low-pass filter with unity gain and $D(s)$ is the inverse of the nominal plant. Note that the order of the low-pass filter $Q(s)$ is three and its parameters depend on the estimation gain of the filter, which means that the design of the disturbance observer is coupled with the design of the state estimator.

The model-based velocity estimator used the following Q-filter in the disturbance observer:

$$Q(s) = \frac{1 + 3\tau s}{1 + 3\tau s + 3(\tau s)^2 + (\tau s)^3}, \tag{5.32}$$

where τ was equal to 0.01. In discrete-time this corresponds to

$$Q\left(z\right) = \frac{1.373 \cdot 10^{-2} z^{-1} - 7.176 \cdot 10^{-4} z^{-2} - 1.215 \cdot 10^{-2} z^{-3}}{1 - 2.715 z^{-1} + 2.456 z^{-2} - 7.408 \cdot 10^{-1} z^{-3}} \tag{5.33}$$

The following high-pass filter:

$$D\left(z\right) = \frac{6.27 - 18.15 z^{-1} + 17.51 z^{-2} - 5.625 z^{-3}}{1 - 2.715 z^{-1} + 2.456 z^{-2} - 7.408 \cdot 10^{-1} z^{-3}} \tag{5.34}$$

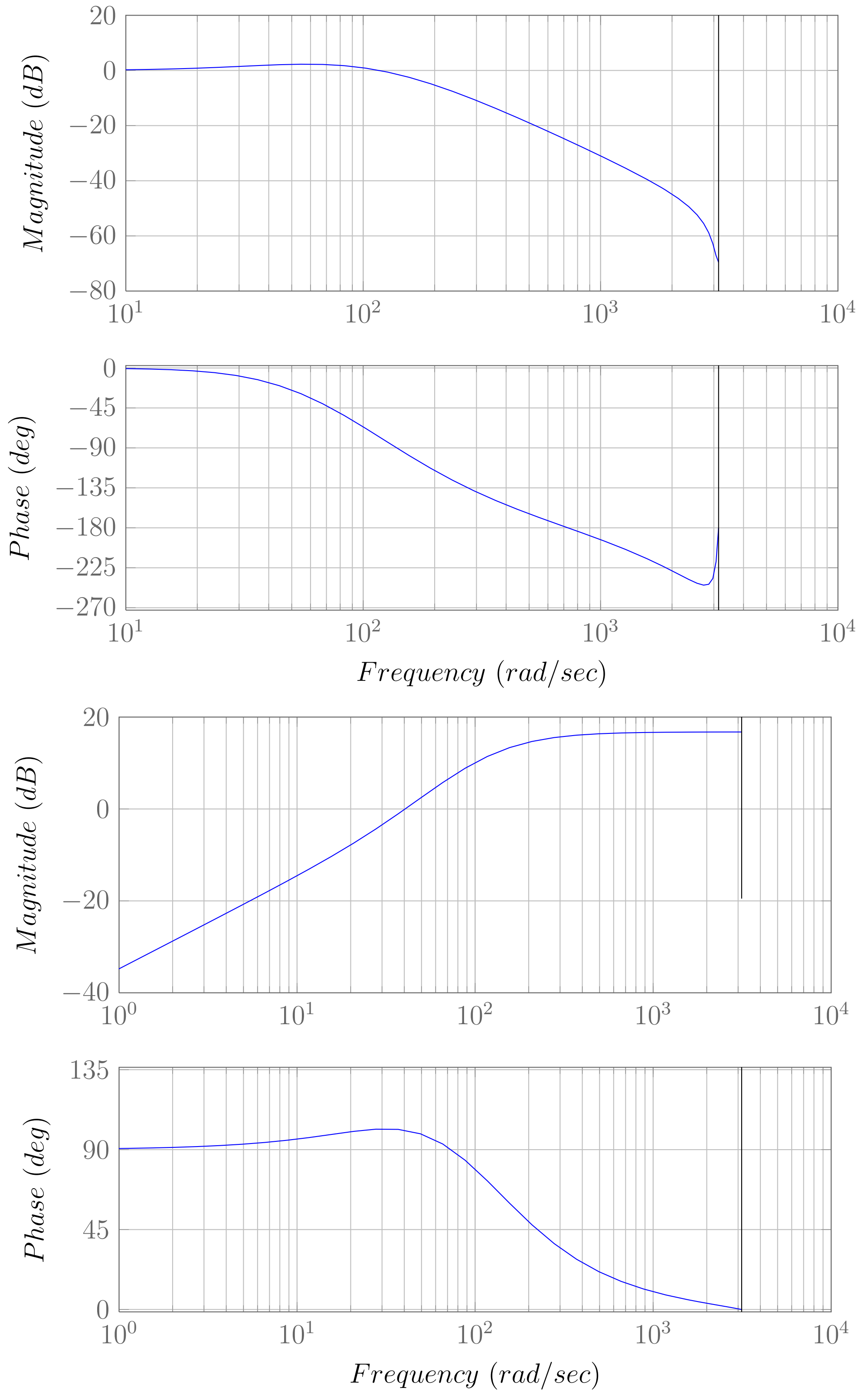


Figure 5.13: Above: Bode Diagram of the transfer function $Q(z)$. Below: Bode's Diagram of the transfer function $D(z)$.

was also implemented.

Regarding the kinematic Kalman filter, the following matrices were found in (4.14) in the discrete-time kinematic model:

$$A_k = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1\times10^{-3} \\ 0 & 1 \end{bmatrix}, B_k = \begin{bmatrix} \frac{T_s^2}{2} \\ T_s \end{bmatrix} = \begin{bmatrix} 5\times10^{-7} \\ 1\times10^{-3} \end{bmatrix}, \quad (5.35)$$

$$Q_k = B_k W_a B_k^T = \begin{bmatrix} 1.25\times10^{-12} & 2.5\times10^{-9} \\ 2.5\times10^{-9} & 5\times10^{-6} \end{bmatrix}, \quad (5.36)$$

where W_a is the variance of accelerometer and is given by its noise variance, assumed to be $W_a = 5\ (rad/sec^2)^2$.
In the same way, the discrete-time algebraic Riccati equation in (4.18) could be solved using the MATLAB instruction:

$$[M, s, e] = dare(A_k', C', Q_k, V);$$

which gave the following one-step-ahead prediction error covariance matrix:

$$M = \begin{bmatrix} 2.0728\times10^{-8} & 1.0412\times10^{-6} \\ 1.0412\times10^{-6} & 1.0204\times10^{-4} \end{bmatrix} \quad (5.37)$$

Using the one-step prediction error covariance, it was possible to calculate the estimation gain in (4.17), giving:

$$F_k = MC^T \left[CMC^T + V\right]^{-1} = \begin{bmatrix} 0.0956 \\ 4.8022 \end{bmatrix}, \quad (5.38)$$

allowing the calculation of:

$$A_{kc} = (I - F_k C)A_k = \begin{bmatrix} 0.9044 & 9.044\times10^{-4} \\ -4.8022 & 0.9952 \end{bmatrix},$$
$$B_{kc} = (I - F_k C)B_k = \begin{bmatrix} 4.522\times10^{-7} \\ 9.976\times10^{-4} \end{bmatrix}. \quad (5.39)$$

In order to analyse the error dynamics equations of (4.12) and (4.19) in more detail using the Kalman filter theory, a qualitative comparison is made by comparing the estimation error covariance of the estimators. The covariance of the estimation error is given by the following equation:

$$Z = M - MC^T \left[CMC^T + V\right]^{-1} CM, \tag{5.40}$$

where M is obtained by solving the discrete-time algebraic Riccati equation, as previously described. Assessment of estimator performance takes into account the (2,2) element of Z $\in \mathbb{R}^{2\times 2}$ which corresponds to $E\left[\tilde{\omega}(k|k)^2\right]$, i.e. the velocity estimation error, under the assumption of the Gaussian white noise property. In this way, it is possible to obtain the standard deviations of the velocity estimation error of the model-based estimator and that of the KKF for various encoder resolutions.

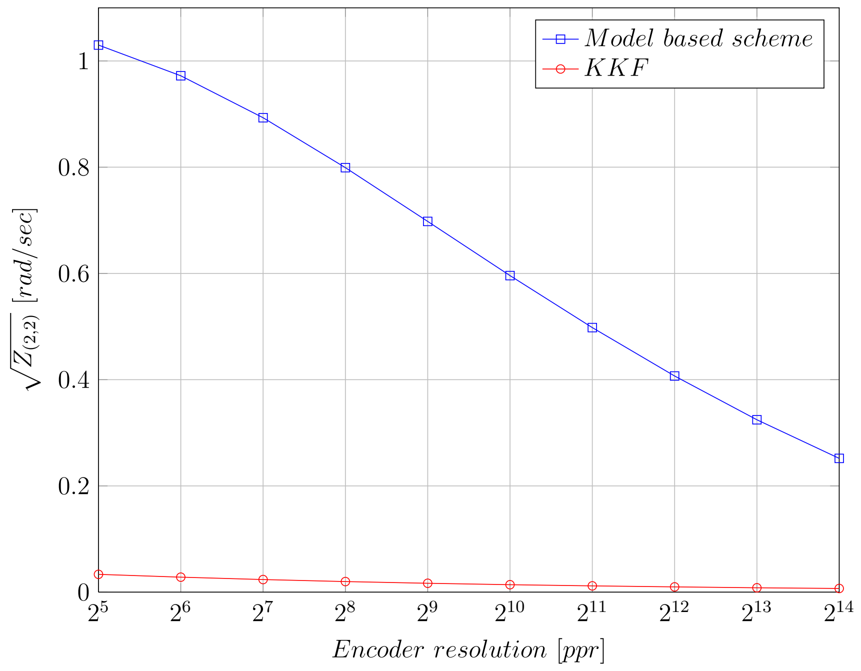


Figure 5.14: Comparison of standard deviations of velocity estimation errors.

Figure 5.14 can be produced using the system data, which shows the standard deviations of velocity estimation errors using various encoder resolutions. The figure shows that the KKF standard deviations of velocity estimation errors are very low even, when the resolution of the encoder is very low. On the other hand, when the resolution of the encoder is lowered, the standard deviations of the velocity estimation errors for the model-based estimator rapidly increase.

The Simulink model used for simulations is shown in Figure 5.15. It includes the PI controller previously described and the two estimators as feedback signals, in order to use the signal estimated by the model-based system, or the KKF, as feedback. This is possible using a switch in the feedback.

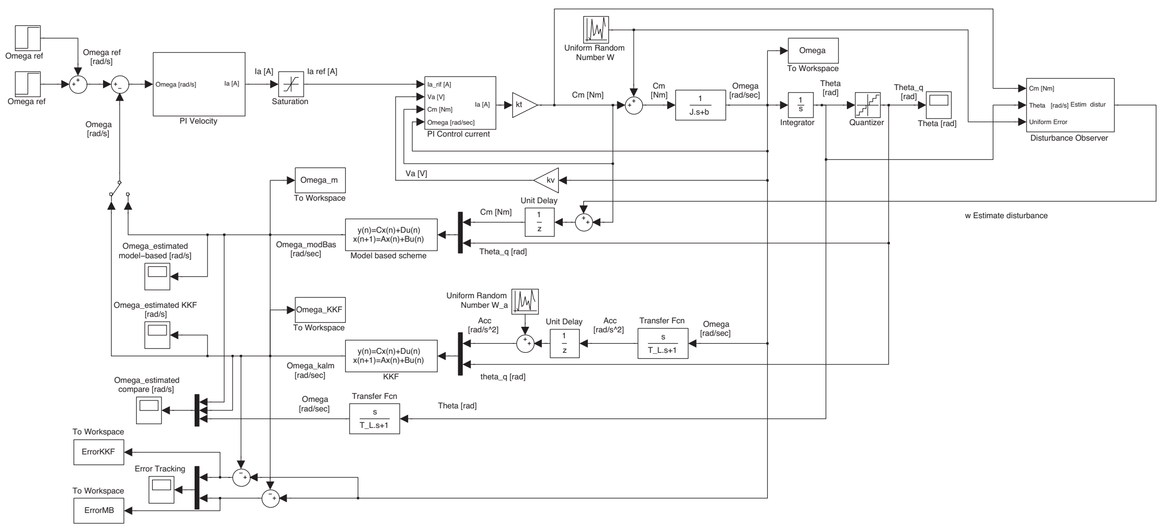


Figure 5.15: Overall Simulink model.

The two models were implemented using the *Discrete State-Space* blocks, inserting the matrices in an appropriate manner. In the model, a disturbance was added to the torque and acceleration signals with the *Uniform Random Number* blocks, in order to simulate the noise present in the real model. Note the addition of the disturbance observer to the right of the diagram in order to implement the Low-pass and High-pass filters, thus estimating the noise present in the torque. In order to derive acceleration from the speed, the following transfer function was used:

$$N(s) = \frac{s}{T_L s + 1}. \tag{5.41}$$

That is simply a high-pass filter with a pole at high frequency, so as to obtain an approximation of the acceleration. The choice of T_L depends on the bandwidth of the closed loop system, because the filter must have a bandwidth greater than this, so that the estimate of the state so obtained is sufficient and does not amplify the measurement noise. Note also the use of the *Quantizer* block to simulate the quantized position read by the encoder.

Figure 5.16 shows simulated response of the system using as feedback signal the speed estimated from the KKF using high resolution encoder, equal to 2^{12} [ppr]. The system input has a step at 3 [sec], which brings the velocity from 4 [rad/sec] to 8 [rad/sec].

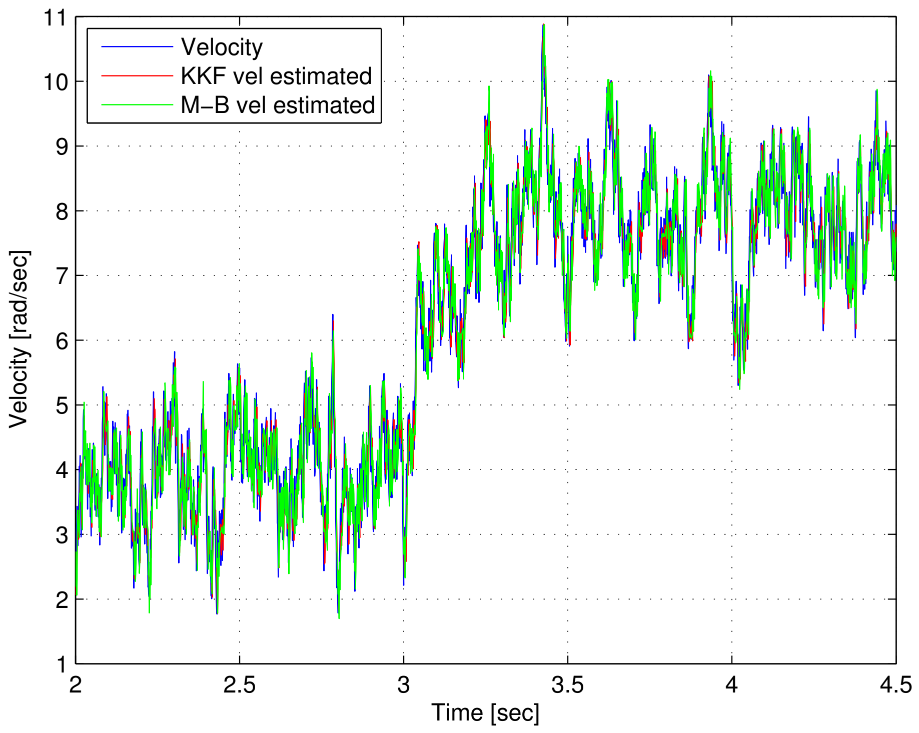


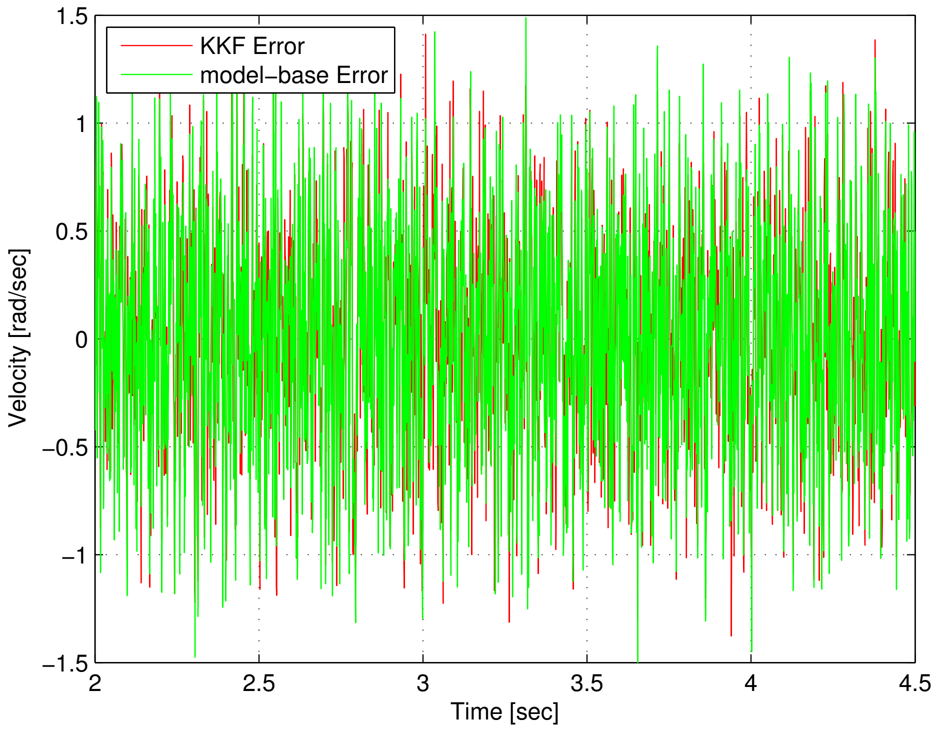
Figure 5.16: Velocity estimation using feedback from KKF, using high resolution encoder.

Both estimated signals follow the monitoring velocity, but the signal estimated by the KKF is more accurate and follows the monitoring signal better. The system response using the velocity estimated by the model-based scheme as a feedback signal is similar to Figure 5.16, and for this reason is omitted.

Instead, it is important to evaluate the difference in the tracking error of the two different responses. In order to do this, Figure 5.17 shows the velocity estimation error using feedback from the KKF and the velocity estimation error using feedback from the model-based scheme, with high encoder resolution. The velocity estimation error in both cases is somewhat lower with the use of KKF. A major difference in the performance can be seen using a low resolution encoder, 2^8 [ppr]. The estimation errors are shown in Figure 5.18.

In both cases, using an encoder with high or low resolution, speed monitoring using an encoder with high resolution is equal to 2^{12} [ppr]. In the latter case, the best performance can be seen from the estimation error, which in this case is much less when using the KKF model. This corresponds to the desirable outcome of a real system and that the project wants to show.

However, it should be noted that various approximations were made in the simulations. By using this approach in the real system, the responses of the system may be quite different. At this point, it was possible to design and test the models implemented in Simulink in the real system.



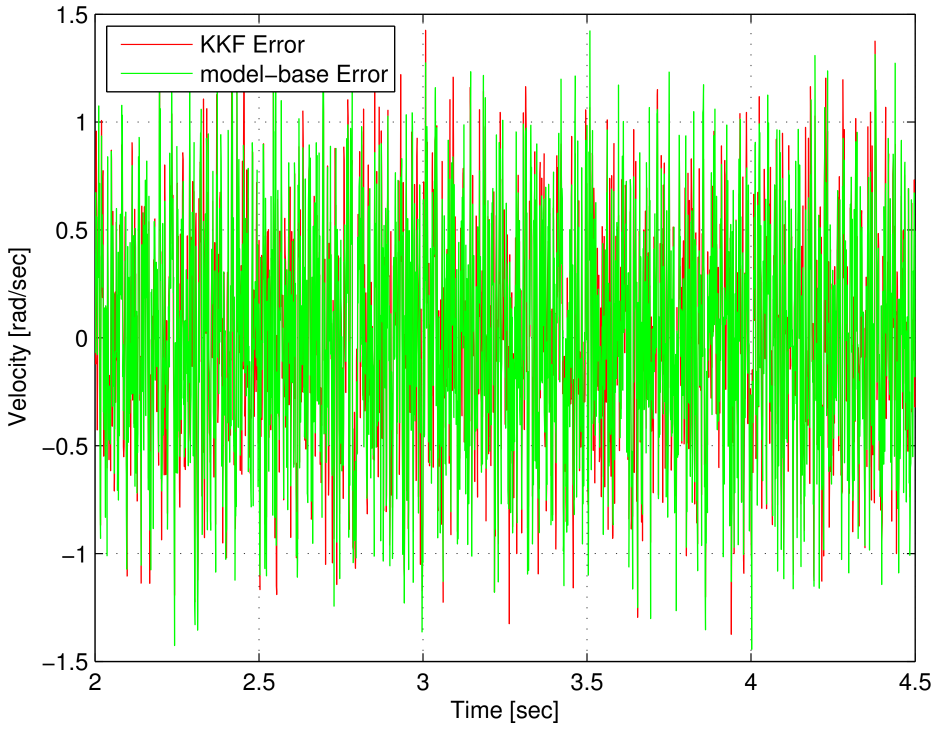
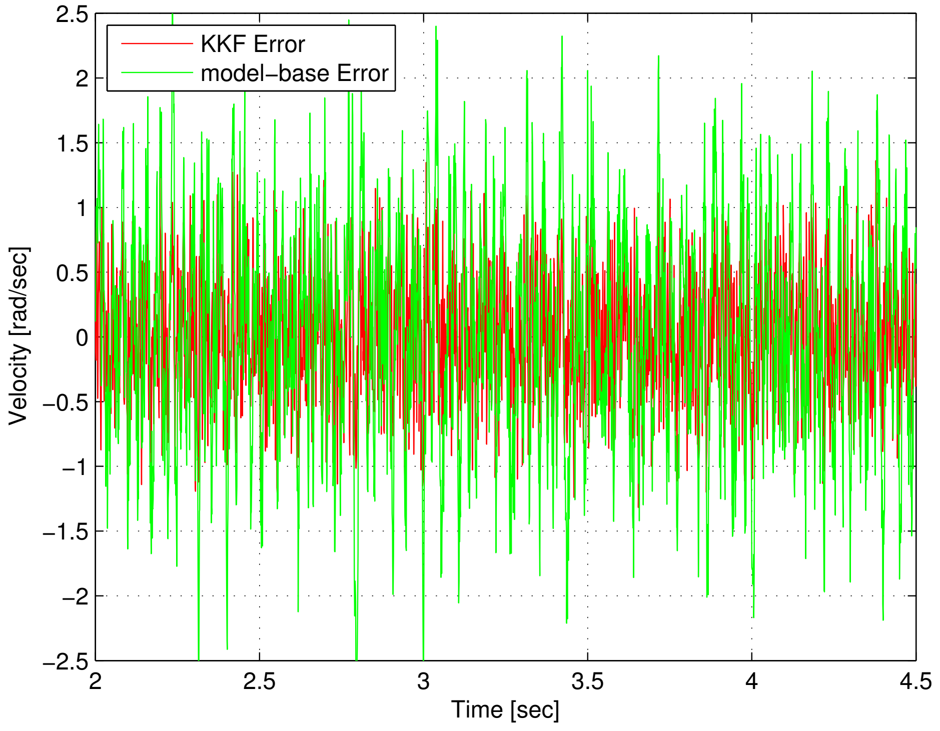


Figure 5.17: Above: Velocity estimation error using feedback from KKF. Below: Velocity estimation error using feedback from model-based scheme. (Error using an encoder of high resolution).



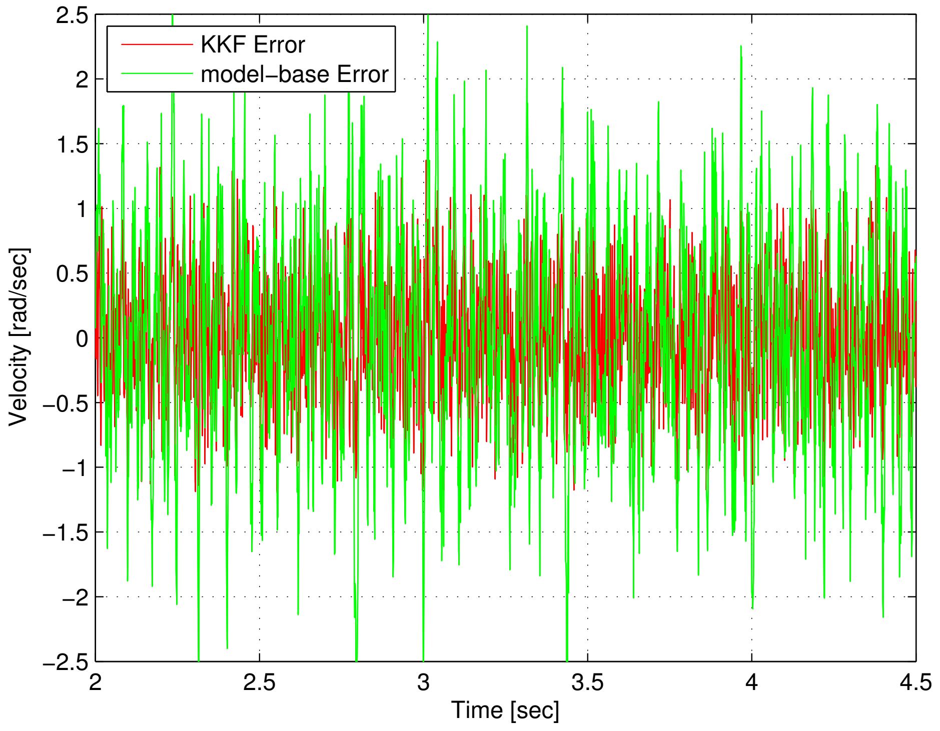


Figure 5.18: Above: Velocity estimation error using feedback from KKF. Below: Velocity estimation error using feedback from model-based scheme. (Error using an encoder of low resolution).

5.4 Design of DSP-Based Controller Language

Firstly, the PI speed controller was implemented in the C language in order to test the controller in a real system using the dSPACE board. The C-Code is reported in Appendix A.1. The commands described in Section 3.5 were used to implement the code.

In the first part of the code implemented two 6th-order, digital, LP Butterworth, filters with flat band characteristics and $f_c = 250$Hz were implemented. The routine is called *filterloop()* and is used to filter the signals provided by the MEMS devices. Signals from two sensors were stored in *ADC_1* and *ADC_2*, respectively. The average signal from the Vero-Board circuit was stored in *analog_avg* and the signal from the *Hoodwin* accelerometer was stored in *Hoodwin*, which was not used in the controller. The average of the signals was also calculated digitally in order to reduce the filtering on the signal used, and stored in *average.*

Using the DS2100 servocontroller, in torque-loop mode, it was necessary to implement the velocity control, as described previously.

The PI controller was implemented with the following lines of code:

```
speed_error = speed_ref - speed;
Pprop = (speed_error)*Kp;
speed_error_int = speed_error_int + (speed_error)*DT*Ki;
current_PI = Pprop + speed_error_int;
```

where the *speed_error* is the difference between the reference signal *speed_reference* and the signal of the measured velocity in the system. The values K_p and K_i are the parameters of the controller, DT is the sampling time and *current_PI* is the output current from the controller.

Note the saturation included in the code, to avoid excessive current demand, which could cause irreparable damage to the inverter and to the motor. Moreover, the saturation was designed taking into account that the current values used in this project were rather low because the project did not demand high performance from the motor.

To test the two estimators, the code in Appendix A.2 was implemented. The appendix includes the code that uses the feedback signal from the KKF. The code that uses the signal from the model-based scheme is very similar with the only change being the feedback signal used. Therefore, the discrete-time model of the two estimators were implemented in the C language using double precision math. With the functions *Initialize_ Model_ kalman()* and *Initialize_ Model_ Base()*, it was possible to initialize the model with the variable *x0_ ka* and *x0_ mb* respectively for the KKF model and the model-based scheme. With the functions Update*Update_ Model_ kalman()* and *Update_ Model_ Base()*, it was possible to update the model. Following each call to the initialization and update functions, the output vector could be accessed with *Output_ Model_ kalman()* and *Output_ Model_ Base()*. If necessary, the state vector could be accessed with *State_ Model_ kalman()* and *State_ Model_ Base()*, respectively for the two models.

Note that the two discrete-time models were implemented including all matrices of the system. The matrix D that in the project was defined as zero. This allows the use of various discrete-time models with a simple change of the values in the matrices and the values of their metrics.

In this case, the original encoder signal, 4096 [ppr], was further quantized to N $= 2^8 =$ 256 [ppr] with the following lines of code:

$$encoder = -(ds1102\_inc(1) * 8388608);$$

$$position\_q = (int)((PPR1/(PPR)) * encoder);$$

$$position\_quan = (position\_q/(PPR1 * 4)) * 2 * 3.1415926;$$

where PPR was equal to 4096 and $PPR1$ was equal to 256.

Using the functions *filterloop2()* and *filterloop3()* both filters $Q(s)$ and $D(s)$ (low-pass and high-pass filter respectively) were implemented as described in the previous section, in order to implement the disturbance observer.

Chapter 6

Results and Analysis

This section gives the results obtained in the real system, using the codes previously described and the dSPACE control board to record the data from the system. The graphs and figures reported were carried out and processed using Matlab.

6.1 PI controller response

The first step was carried out to test the response of the PI speed controller in the real system. This was possible with the use of the code in A.1, which produced the response shown in Figure 6.1. Figure 6.1 shows the system response using a step input of height 20 [rad/sec]. In this case the test was conducted using the original parameters calculated in (5.23). The system response had a rise time of about 0.1 [sec] (i.e. from 10% to 90% of the step height) and an overshoot of 25% of the step height.

In order to reduce the overshoot, the integral gain was slightly decreased by bringing it to a value of $K_{I\omega} = 0.2$ from the original $K_{I\omega} = 0.2152$. By decreasing the integral gain it was possible to obtain a lower overshoot and a larger phase margin, but ,in contrast, a greater steady-state error could be present. Figure 6.2 shows the response of the system using this change to the parameter $K_{I\omega}$ and a step input of height 50 [rad/sec].

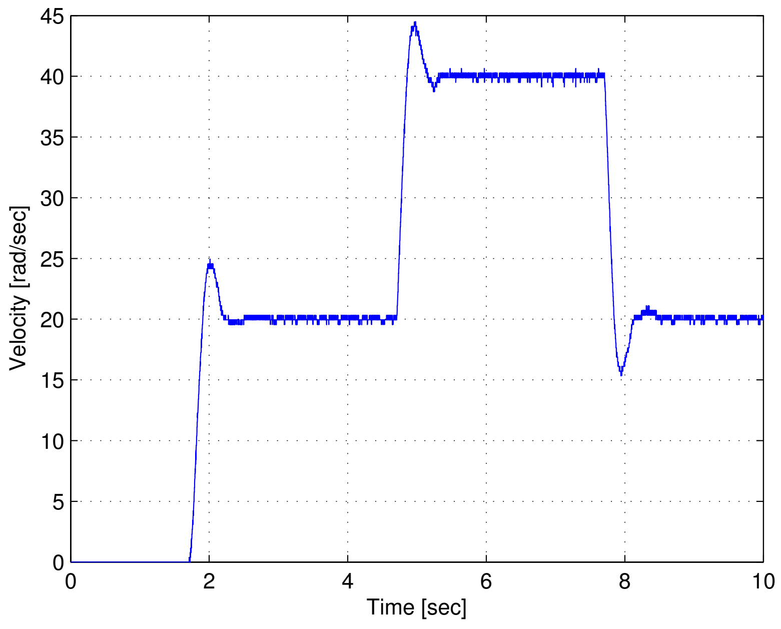


Figure 6.1: PI controller response using the original parameters $K_{P\omega} = 0.0086$ and $K_{I\omega} = 0.2152$.

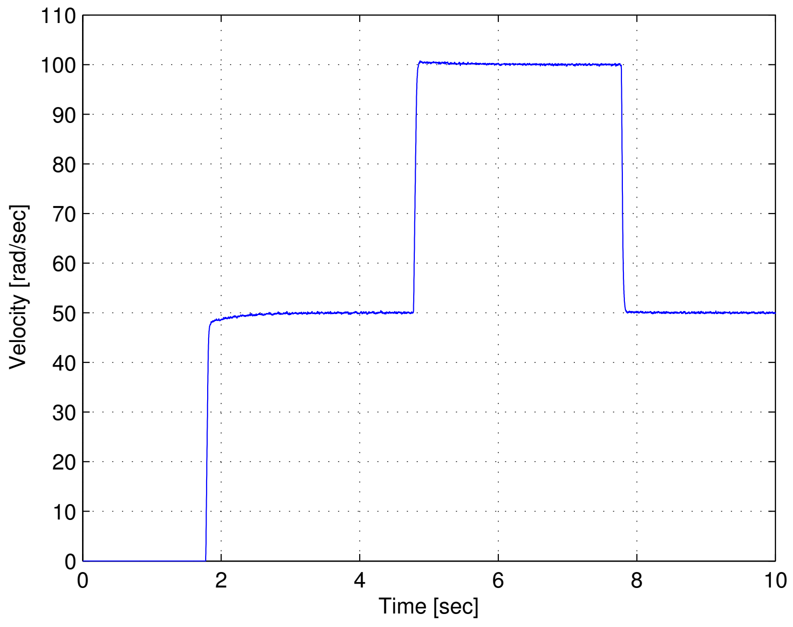


Figure 6.2: PI controller response using the parameters slightly modified, $K_{P\omega} = 0.0086$ and $K_{I\omega} = 0.2$.

The system response had a rise time of approximately 0.08 [sec] and did not overshoot. The signal response followed the reference signal perfectly when the integral gain was decreased.

Therefore, the controller designed in the previous section, with slight modifications, had excellent performance in the real system.

However, its performance could be changed by varying the gains. In subsequent tests the PI control parameters were as described in this Section (Figure 6.2).

6.2 Results for KKF and Model-based schemes

At this point, the estimators were tested in the real system using the C-code shown in A.2. Firstly, the estimators were tested using the original encoder signal, N = 4096 [ppr], both for the inputs of the estimators and for the monitoring signal. In fact, the successive difference of the original 4096 [ppr] encoder was used for real velocity monitoring and also to provide a feedback signal.

Therefore, the resolution of the monitoring velocity is given by:

$$\text{Resolution} = \delta\omega = \frac{\delta\vartheta}{T_s} = \frac{2\pi}{2^{12} \cdot 0.001} = 1.5340 \;\; [rad/s] \tag{6.1}$$

This means that the monitoring velocity is guaranteed to be accurate within this bound.

Using this value of quantization, several tests were carried out on the system, in order to determine the best value of the input variances W and W_a. The diagrams in Figure 6.3 and 6.4 show the maximum velocity estimation error using the model-based scheme and KKF respectively, by varying the values of input variance W and W_a. Note that the best value for W is 7.5×10^{-3} $(Nm)^2$ with maximum velocity estimation error of 1.2 [rad/sec], and the best value for W_a is 5 $(rad/s^2)^2$ which corresponds to a maximum velocity estimation error of 0.7 [rad/sec].

In this case the following relation is obtained:

$$J^2 W_a = 2.1841 \times 10^{-7} \ll 7.5 \times 10^{-3} = W$$

which means that the amplitude of F_k is significantly smaller than the one of F_m. So using the original signal encoder input parameters of the variances are not different from those used in simulations. Therefore, in order to see the difference in amplitude of the two gains of the estimators, the matrices given in (5.29) and (5.38) can be compared. At this point, the estimators were tested in the real system using the estimated velocities as feedback signals to the controller.

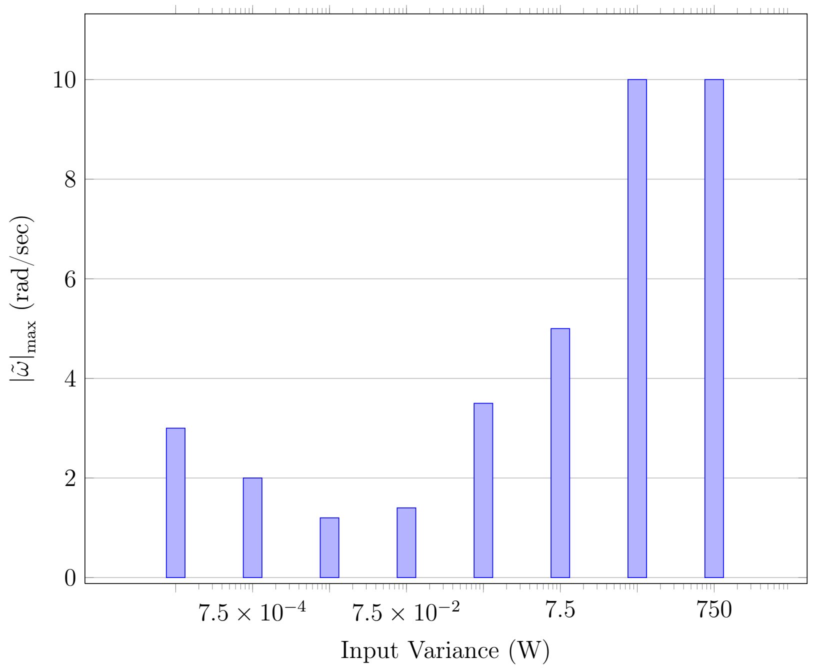


Figure 6.3: Velocity estimation error as a function of assumed input variance value for model-based scheme with high resolution encoder.

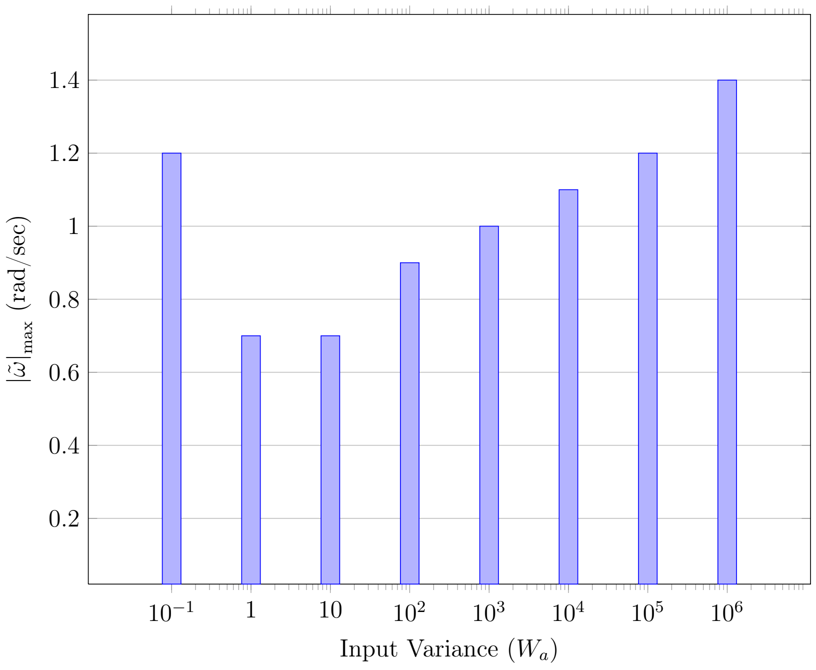


Figure 6.4: Velocity estimation error as W_a varies (KKF with high resolution encoder).

Figure 6.5 shows the velocity tracking with estimated velocity, using the velocity feedback from the model-based scheme. The figure shows the response to a step input to the system with height of 4 [rad/sec]. In fact, the velocity is set constant at 4 [rad/sec] initially and then changes to 8 [rad/sec].

The estimated velocities from the two estimators can be considered quite similar, with a slightly greater uncertainty to the signal estimated by the model-based scheme. In order to better see this response, a magnification where there is a step was carried out and is shown on the next page (in Figure 6.6 Above). In Figure 6.6 Below the relative estimation errors are shown. In this way, the two responses can be seen in more detail and their estimation errors can be compared. As can be seen, the signal estimated by the KKF has lower estimation error, with a peak equal to 0.4 [rad/sec]. However, even the velocity signal estimated by the model-based scheme has not very high errors, having a peak of about 0.7 [rad/sec].

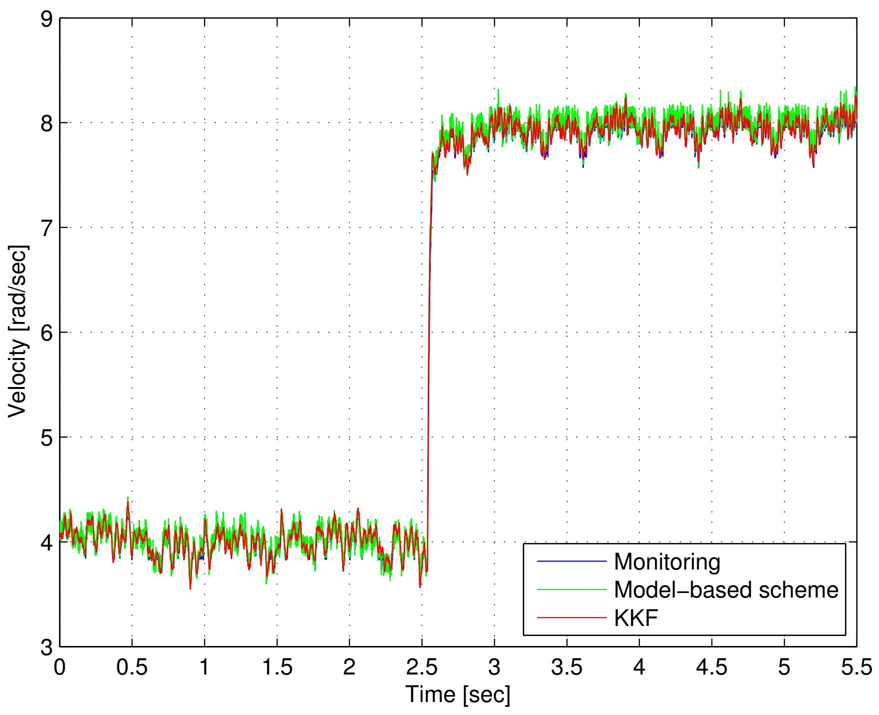
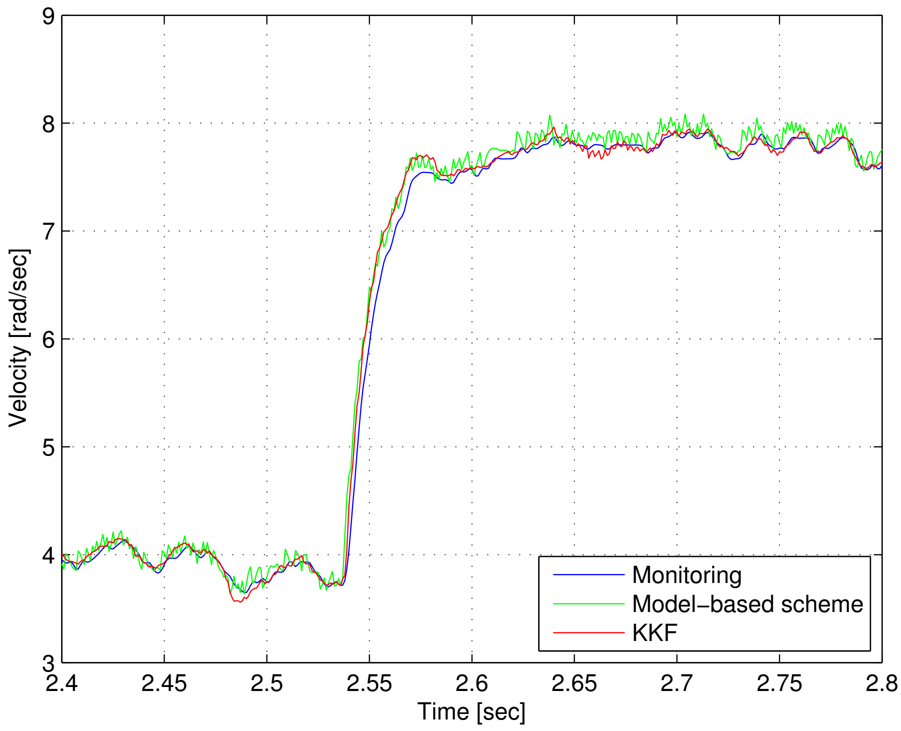


Figure 6.5: Velocity tracking with estimated velocity. Velocity feedback from the model-based scheme using high resolution encoder.



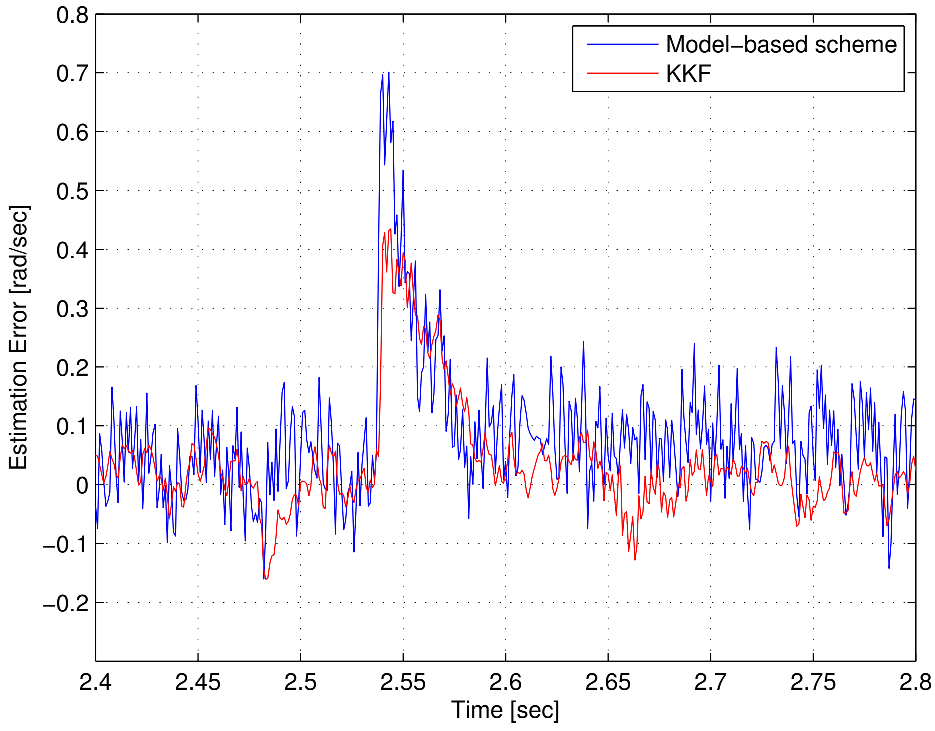


Figure 6.6: Above: Velocity tracking with estimated velocity. Below: estimation errors. Velocity feedback from the model-based scheme.

So both estimated velocities are within the bound imposed by the resolution of velocity monitoring in (6.1). Figure 6.7 shows an enlarged view of the velocity tracking. Note how there are slight fluctuations in the estimated velocity from the model-based scheme. These are due to the cogging force, but are not in the speed estimated by the KKF which perfectly follows the monitoring velocity.

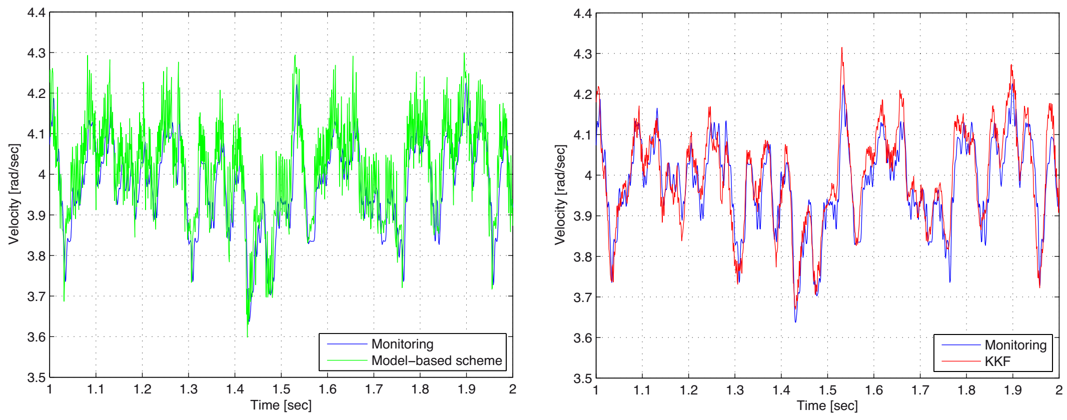
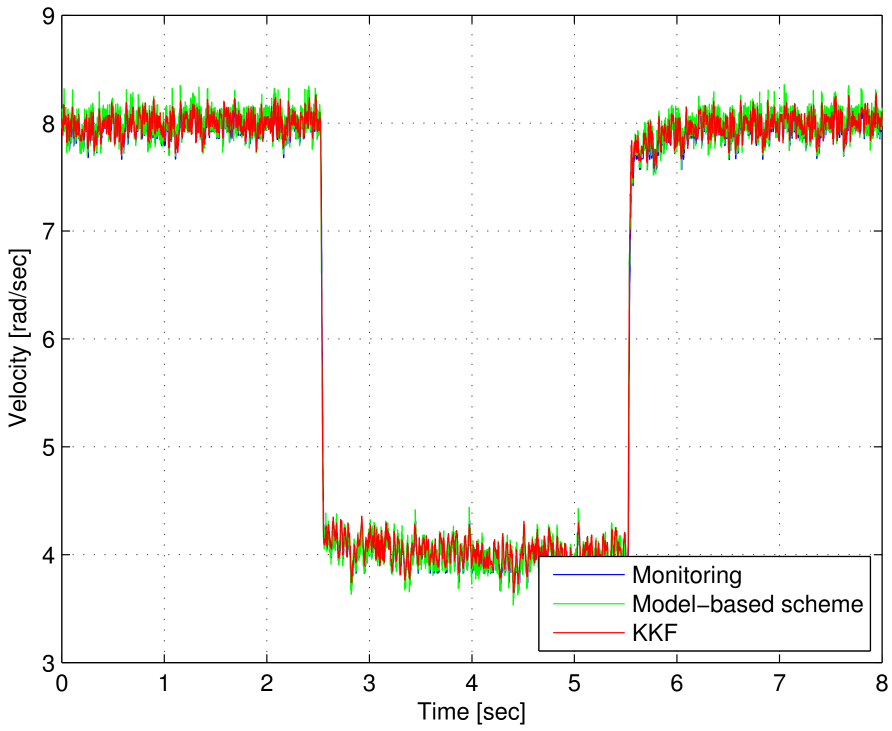


Figure 6.7: Enlarged view of the velocity tracking.

The system response was also tested when using the signal estimated by the KKF as feedback signal to the controller. Figure 6.8 shows the velocity tracking with estimated velocity feedback from the KKF. In this case, the same previous descriptions can be done in this case, as for the model-based scheme. Because the velocity estimated by the KKF has a lower estimation error, it can follow the monitoring signal better. In this case, a slight offset from the signal, estimated by the KKF, may be noticed as can be seen from the estimation errors, which are due to a slight error in the Zero Offset of the output signal from the accelerometers. This salient point will be discussed in detail below.



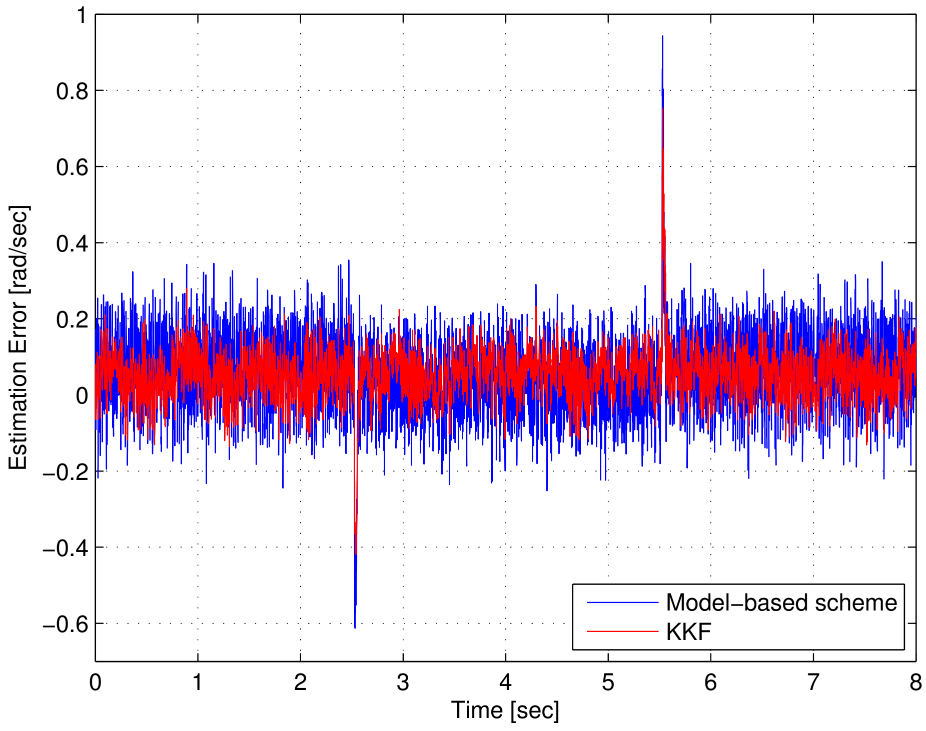


Figure 6.8: Above: Velocity tracking with estimated velocity feedback from the KKF, using high resolution encoder. Below: estimation errors.

The estimators were here tested using a signal from the low resolution encoder (with N $= 2^8 = 256$ [ppr]). In order to get a signal from the encoder with low resolution, the original signal of the encoder is further quantized to a coarser quantization level. The resolution of the encoder in this case is for the quantized version of the original 4096 [ppr] encoder by a factor of 2^4. If the velocity is estimated using a difference of successive encoder counts, the velocity resolution becomes:

$$\text{Resolution} = \delta\omega = \frac{\delta\vartheta}{T_s} = \frac{2\pi}{2^8 \cdot 0.001} = 24.5437 \text{ [rad/sec]}$$

which is even larger than the reference velocity. Again, several experiments were conducted in order to determine the best value of the input variances W and W_a, using this value of quantization. The diagrams in Figure 6.9 and 6.10 show that the maximum velocity estimation error using the model-based scheme and KKF respectively, by varying the values of input variance W and W_a.

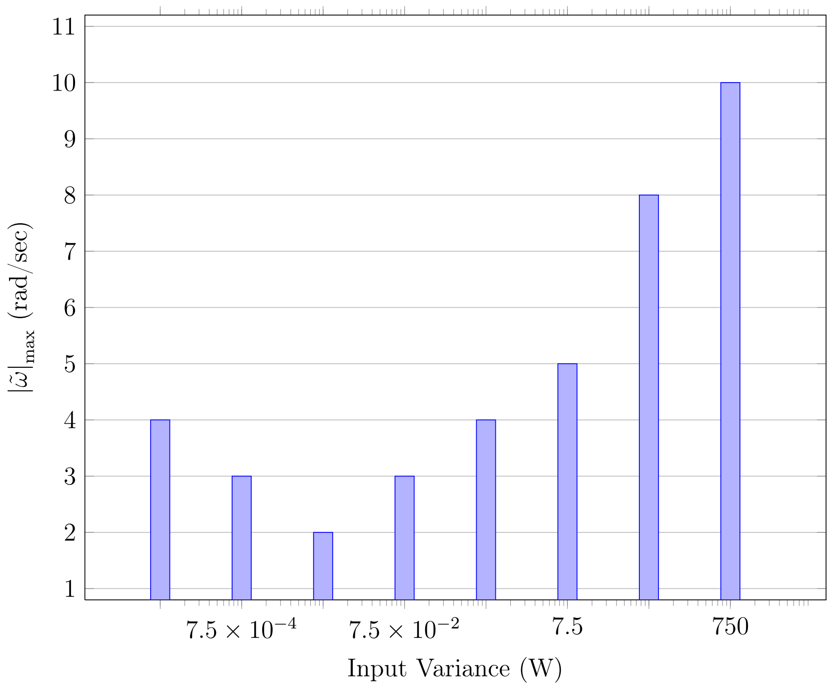


Figure 6.9: Velocity estimation error. Model-base estimator with low resolution encoder.

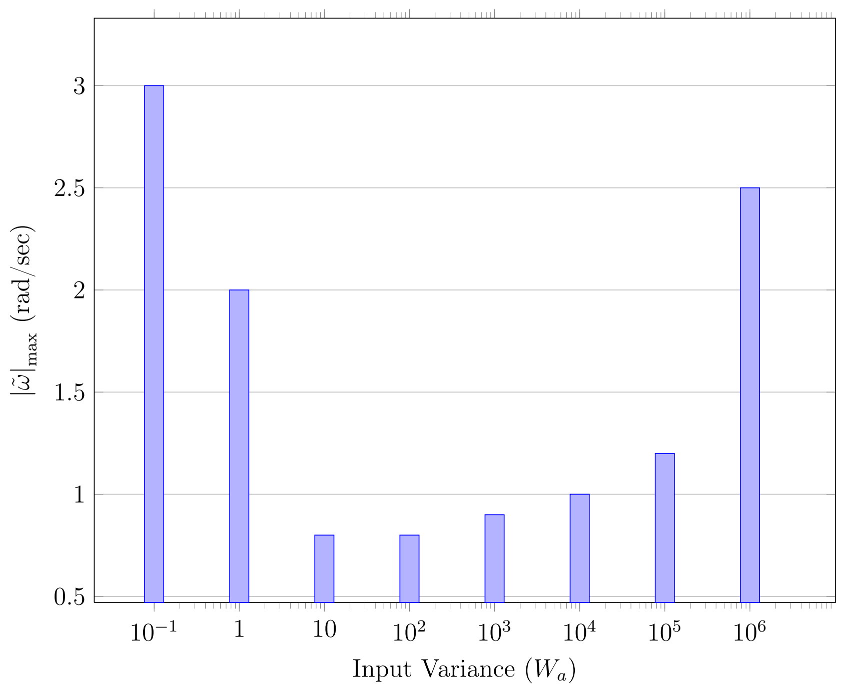


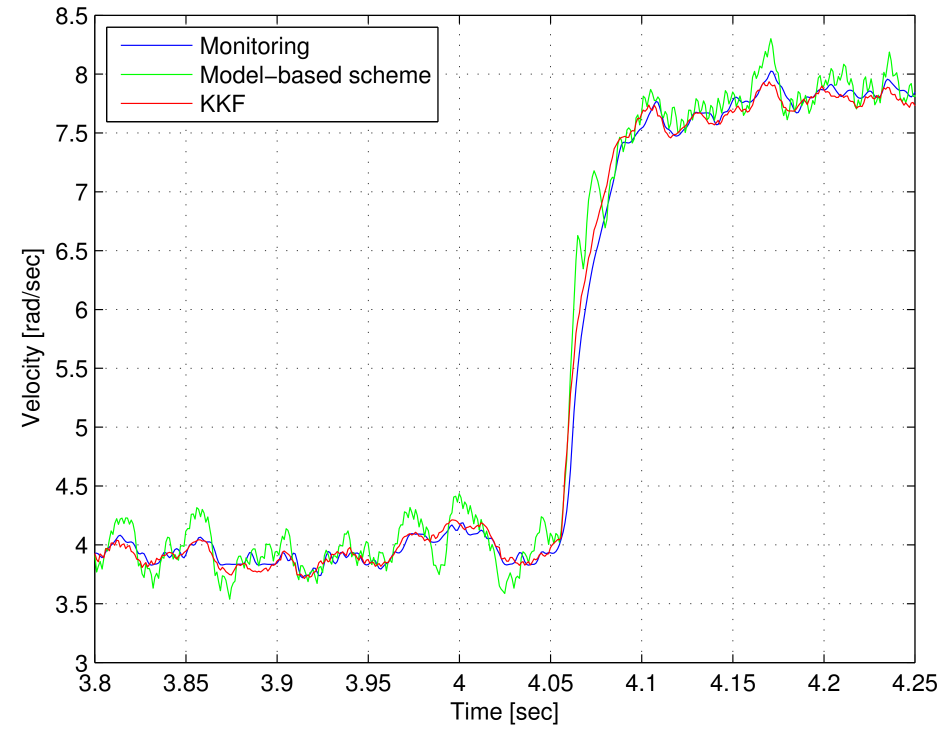
Figure 6.10: Velocity estimation error. Kalman estimator with low resolution encoder.

The best value obtained for the input variance for the model-based scheme was the same as the one obtained in the design. For the input variance for the KKF, the best value was chosen as $W_a = 10\ (rad/s^2)^2$. As can be seen from the diagrams, the KKF model is less sensitive to changes in input variance in comparison to the model-based scheme. This will be noted below.

By using this encoder signal, the matrices of the models have changed because the value of the variance V was varied. The gains of the estimators found using these parameters are as follows:

$$F_m = \begin{bmatrix} 0.2422 \\ 33.5329 \end{bmatrix}, F_k = \begin{bmatrix} 0.0294 \\ 0.4397 \end{bmatrix}. \tag{6.2}$$

Figures 6.11 and 6.12 show the responses of the estimators using feedback from the model-based scheme and the KKF respectively, using the encoder signal quantized with N $= 2^8$ [ppr].



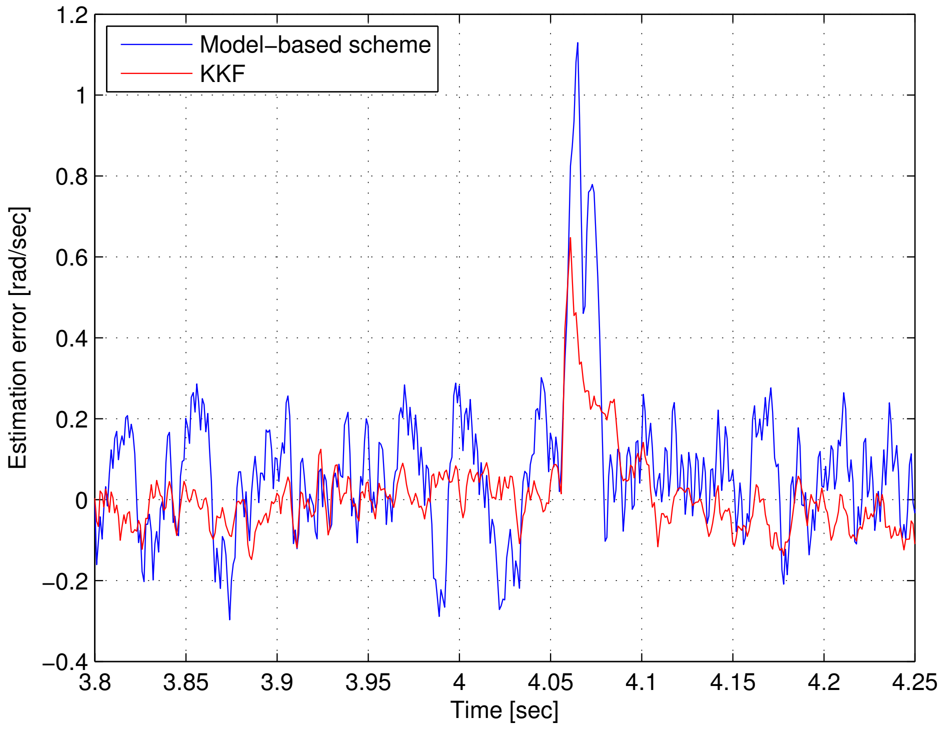
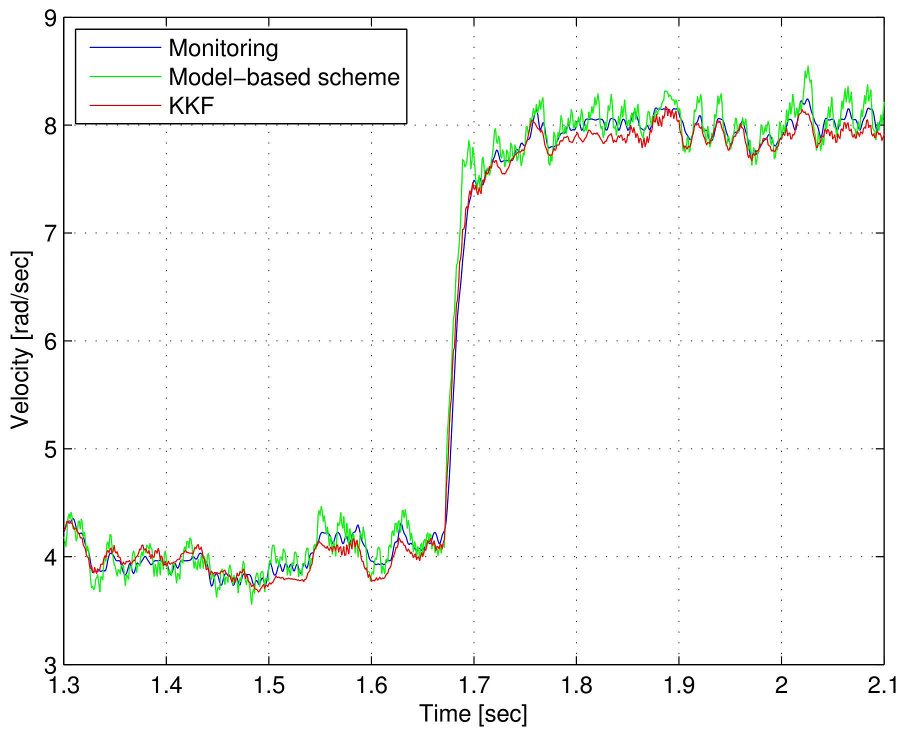


Figure 6.11: Above: Velocity tracking with estimated velocity. Feedback from the model-based scheme using low resolution encoder. Below: Estimation errors.



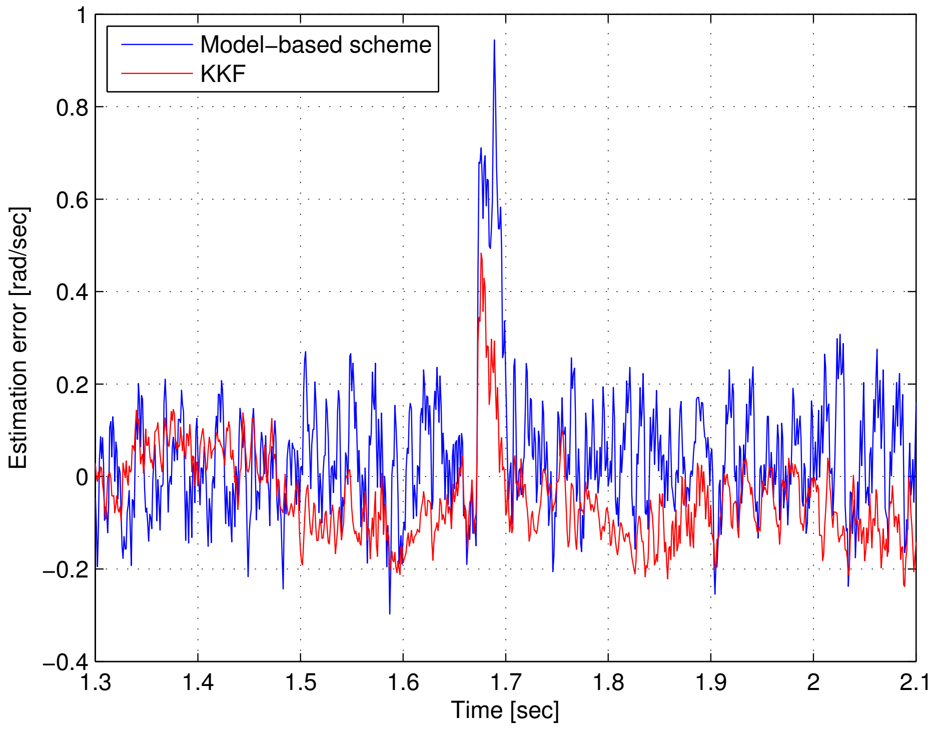


Figure 6.12: Above: Velocity tracking with estimated velocity. Feedback from the model-based scheme using low resolution encoder. Below: Estimation errors.

As can be seen from the figures, the KKF model perfectly follows the monitoring signal and has an estimation error less than the model-based scheme, both with feedback from the model-based scheme or the feedback signal from KKF. In this case, taking a value slightly greater of the variance W_a, implies a slightly faster estimator dynamics. In fact, a faster estimation dynamics implies a higher gain of the estimator, and this makes the KKF more sensitive to the quantization effect. In this case, as reported in [1], this does not imply a lower phase delay but it means that the estimation is more dependent on the position signal which is corrupted by the quantization error and less dependent on the acceleration measurement. This slight modification may be caused by increased noise of the acceleration measured by the MEMS devices. As will be described below, this can be caused by a misalignment of these devices. Therefore, if there is a good acceleration measurement as input to the KKF, the estimation can be more accurate, even with slower estimation dynamics.

In order to better understand this, the equations of estimation error dynamics must be considered, the equations are given by (4.12) and (4.19). If the parameter of viscous damping B is small, the system matrices in (4.7) and (4.14) are related by the following relationship:

$$A_k \sim A_m, B_k \sim JB_m. \tag{6.3}$$

In this way, using the equations (4.12), (4.19) and (6.3) together, the performance of the model-based estimation method and the KKF can be compared by examining the effect of $\tilde{w}$ and w_a, respectively. Therefore, if W and J^2W_a are close to each other, the closed-loop eigenvalues of A_{mc} will be close to those of A_{kc}. So their transfer functions from the input perturbation terms $\tilde{w}(z)$ and $w_a(z)$ to the velocity estimation errors $\tilde{\omega}_m(z)$ and $\tilde{\omega}_k(z)$, are related by the following relationship:

$$\frac{\tilde{\omega}_m(z)}{\tilde{w}(z)} \sim \frac{1}{J}\frac{\tilde{\omega}_k(z)}{w_a(z)}, \tag{6.4}$$

where

$$\begin{aligned} \frac{\tilde{\omega}_m(z)}{\tilde{w}(z)} &= C_\omega(zI - A_{mc})^{-1}B_{mc}, \\ \frac{\tilde{\omega}_k(z)}{w_a(z)} &= C_\omega(zI - A_{kc})^{-1}B_{kc}. \end{aligned} \tag{6.5}$$

with $C_\omega = [0 \;\; 1]$. Accordingly, the closed-loop poles of the KKF can be assigned to produce significantly slower estimation dynamics than the model-based estimator as long as the magnitude of the variance w_a is sufficiently smaller than the one of $(1/J)\tilde{w}$. As described above, slower estimator dynamics means a smaller estimator gain and then the KKF less sensitive to the quantization effect. As noted, the magnitude of $(1/J)\tilde{w}$ is usually significantly larger than that w_a unless it has an unreasonably high noise level or misalignment.

By varying the parameters of the variances W and W_a, choosing respectively values of 100 $(rad/s^2)^2$ and 7.5×10^{-2} $(Nm)^2$, the response in Figure 6.13 was obtained, where the feedback signal was obtained using the model-based scheme.

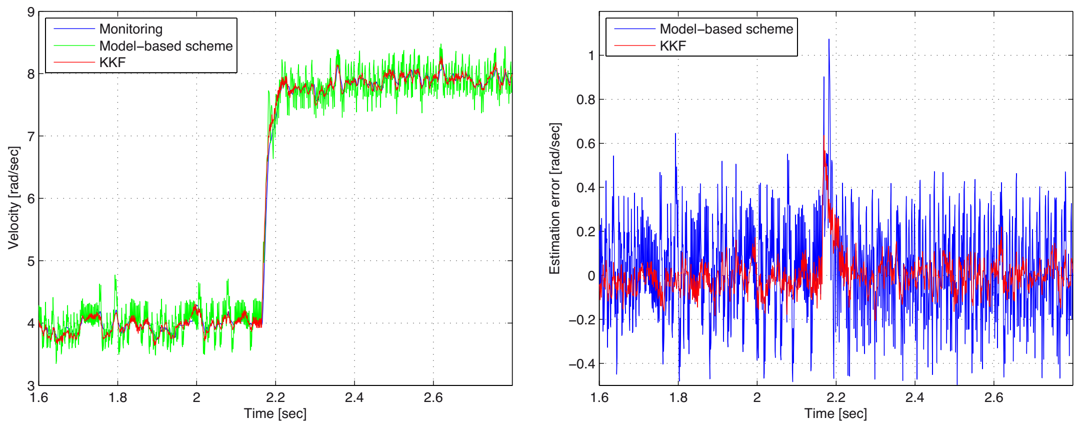


Figure 6.13: Left: Velocity tracking with estimated velocity. Feedback from the model-based scheme using low resolution encoder. Right: Estimation errors.

The estimated signal by KKF still perfectly follows the signal monitoring, but the estimated signal by the model-based scheme has many fluctuations and hence a greater estimation velocity error. This confirms that the sensitivity of the parameters changing for the KKF compared to the model-based scheme.

Thanks to the results obtained from various experiments, it was possible to know the advantage and benefits of using accelerometers with the use of a kinematic model. In fact, the use of the kinematic model together with the use of accelerometers has made the velocity estimate very insensitive to interference and insensitive to quantization levels of the encoder. The KKF has obtained excellent results for both the model that uses a high resolution encoder and for the experiments that use a low resolution encoder, confirming its superiority.

However, some important consideration must be taken into account. In regard to the model-based scheme, the implementation of the disturbance observer is not of simple design, and the estimated noise is always very difficult to approximate to the real noise present in the system. Secondly, the estimation error due to the encoder quantization effect q_θ is amplified by the magnitude of the observer gain. This is related by the disturbance estimation error W, which is a design parameter to be selected, that is not a simple task.

An important aspect analyzed by the various experiments is the importance of a perfect alignment of MEMS devices mounted on the outside of the disk. In the project [7] and in these experiments, the double sided tape was used to fix the devices on the disk. This has led to the displacement of the devices from their original position and hence a wrong reading of the acceleration. The error due to misalignment is described in detail in section 3.3.1. Even a small shift of the devices, due to motor torque, causes a change on the offset of the device and in some cases an increase in signal noise. All this contributes to an increase in the error of the measured acceleration and therefore a worse estimate of the KKF model, which receives this signal in input. In fact, with an error of the acceleration signal, in some cases it was necessary to increase the value of the variance W_a in order to make the estimator from KKF less sensitive to acceleration and more dependent to the position, which signal is corrupted by the quantization error.

Figure 6.14 shows an example of misalignment of the sensor. In this case, the system had a step input that brought the velocity from 50 to 200 [rad/sec], using high accelerations. In Figure 6.14 the output signal from the devices is shown,

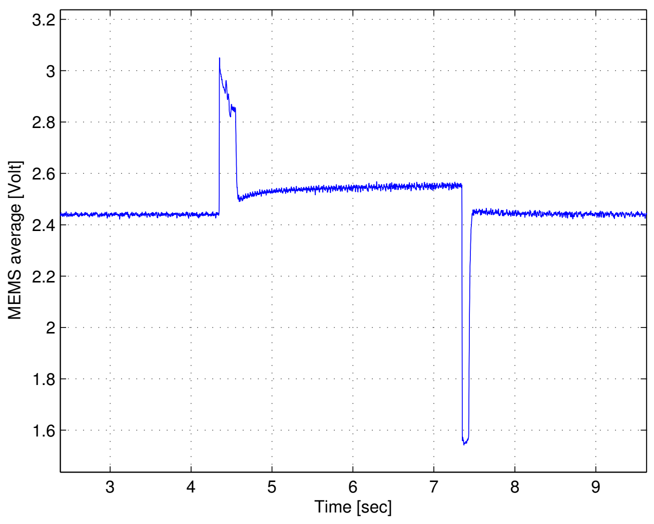


Figure 6.14: Signal average of MEMS devices misaligned.

mediated through the Vero-board circuit. Note how the offset changed when the speed was changed. The sensors were found to move slightly with the change of speed. However, this was just an example of a degenerative case, but even if the sensor was fixed more accurately, a small mistake might always be present. Therefore, it is important to set the devices aligned as accurately as possible, using another stronger and less flexible method of fixing the MEMS sensors to the disk.

Moreover, other aspects must be considered. The power supplied to the accelerometers, though it is assumed constant, always exhibits slight fluctuations around the desired value of 2.5V due to noise and disturbances. Therefore, the Zero Offset of the output signal of the accelerometers may vary slightly. Due to the high amplification applied to the MEMS output, a small offset error can cause a large error for the calculation of the angular acceleration. This is another aspect that must be improved for an excellent use of the accelerometers. This could be solved using the digital output of the sensors, which would not have these disorders.

Finally, the various delays introduced by the DSP must be taken into account. In spite of the high working speed of the board, a small delay in the acceleration signal is always present and therefore may cause errors in estimation of the velocity signal.

Conclusion

The project examines the advantages and benefits of the use of accelerometers in applications where precision work with relatively low speeds is required. An important aspect of the use of accelerometers with the kinematic model is to make the estimated velocities insensitive to perturbation and noise in the system and make it insensitive to different quantization levels of encoders.

The superiority of using the KKF and the acceleration sensors was confirmed by the results obtained from various experiments and analysis of the system. This is interesting from the industrial point of view, as the increasing performance demands of motion control systems require more and more precise instruments, while trying to keep costs low. In this case, the cost of using one or more accelerometers is significantly lower than using a high resolution encoder.

Despite the excellent results obtained, important considerations were discussed with regard to the analog output signal of the acceleration. Indeed an issue which arises from these devices using a analog output, is that the output signal has a DC offset, about which the signal oscillates while the device is experiencing a constant velocity. This offset is dependent on the supply voltage and, if there is a small misalignment of the device, the velocity at which device is traveling. This leads to a worse measurement of the acceleration and consequently a worse estimate from the KKF. Therefore, a better fixing of the devices on the disk must be done and the use of their digital output must be taken into account, in order to have an excellent measure of the acceleration.

LogBook

6.3 Week 1

Read final year project report entitled "Development of Rotary Accelerometer using MEMS Accelerometer" in [7]. This report describes in detail the implementation of the MEMS accelerometer system, assembled and tested by other students.

Read the main parts of the Moog manual [11] to get to know the operation and installation guide of "WinDrive". "Windrive" makes it possible to operate the motor with speed control or torque control.

I also read the articles [21], [22], [23] and [24] in order to know in more detail the use the estimators that use the acceleration signal, the use of the disturbance observer and the compensation of friction.

Read the various C codes written by other students, to acquire data using a DSP board. When I received the "WinDrive" guide, I installed it and tested it to learn all the details of the operation.

Once the motor was able to work, I tested the system using an oscilloscope. I conducted several tests on the system, in order to check that everything was correct.

Many tests consisted of acquisition data through the DSP and verifying them using MATLAB, in order to become familiar with the overall system. For example, the data recorded by the dSpace system for the individual MEMS is illustrated in Figure 6.15. The figure also shows the avareged signal of the MEMS.

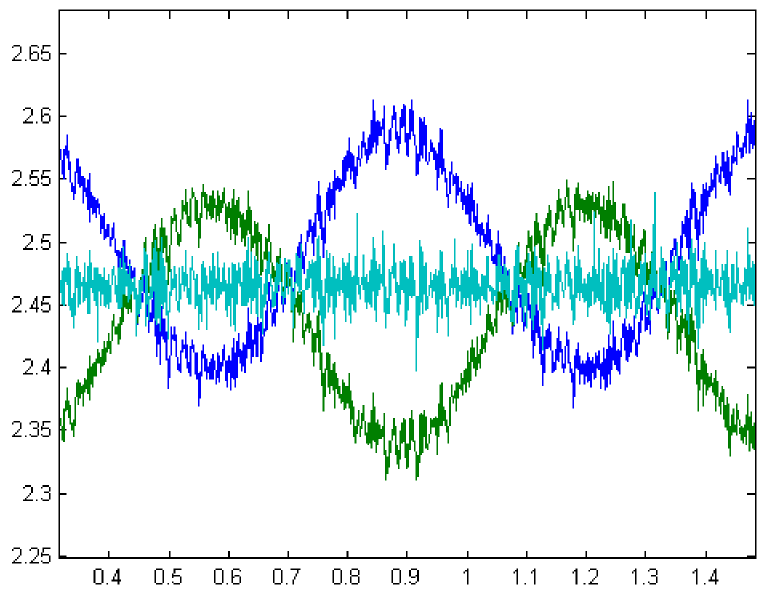


Figure 6.15: MEMS signal at 10 rad/s.

6.4 Week 2

Read the main parts of the DSP manual in [25],[26],[27],[28],[29], to learn how to interface a new board. MATLAB and Simulink can be used for real-time data acquisition which may be useful in carrying out the project.

Looked for accurate data for the motor and driver on the internet, and subsequently found in the data sheet [10], in order to implement the model in Matlab-Simulink.

Designed the Simulink model and then proceed to design the PID controller, but for the moment it remains to be completed in detail.
For example, the Simulink model of the velocity tracking experiment with the model-based scheme is illustrated in Figure 6.16. At the moment the model is temporary and must be improved.

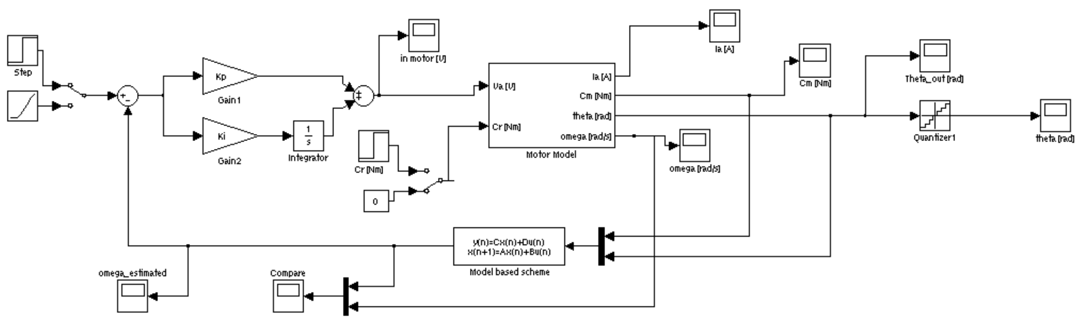


Figure 6.16: Block diagram of the velocity tracking experiment with the model-based scheme.

Figure 6.17 shows in detail the block "Motor-Model", which is the motor model used in simulation. The motor has been brought to the workshop so we can install an additional encoder on the end of the shaft.
In Figure 6.18 the motor can be seen: on the left without encoder and on the right with the encoder.

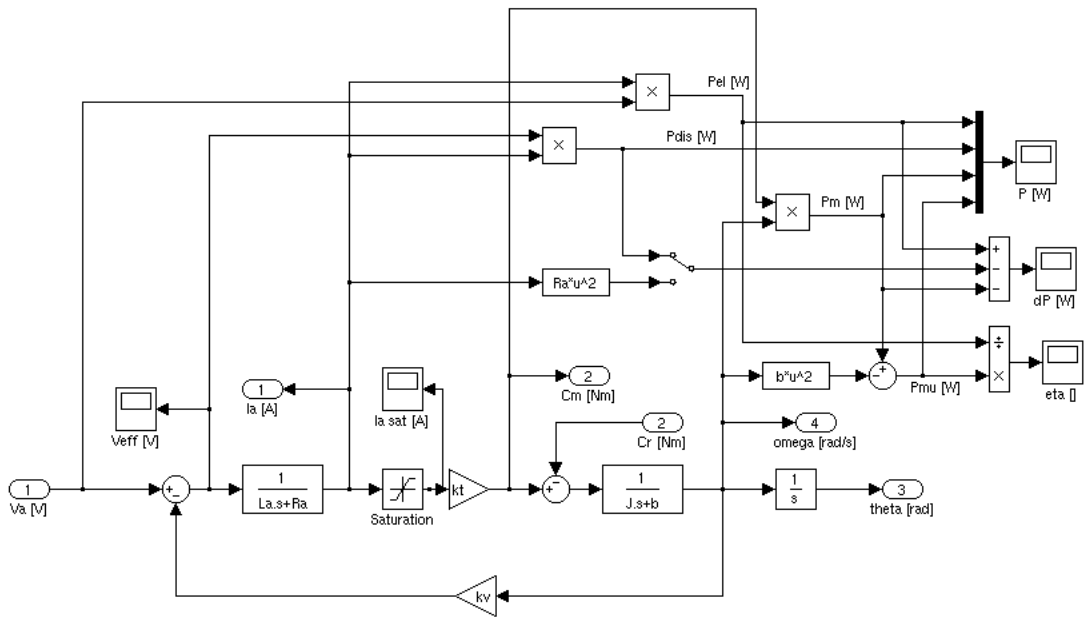


Figure 6.17: Simulink scheme of the motor model.

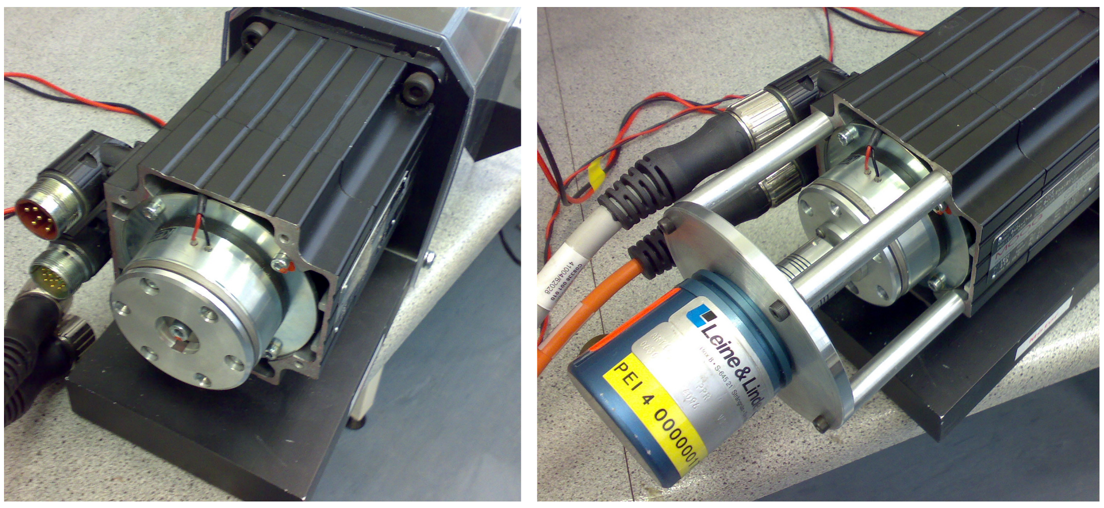


Figure 6.18: Right: Motor without encoder. Left: Motor with encoder.

6.5 Week 3

This week I have implemented a scheme in order to control the motor speed.
To control the speed of the model two controllers are required, one for current control and one for speed control.

Figure 6.19 shows the complete model with two controllers, the velocity feedback from the model-based scheme and the velocity feedback from the KKF.

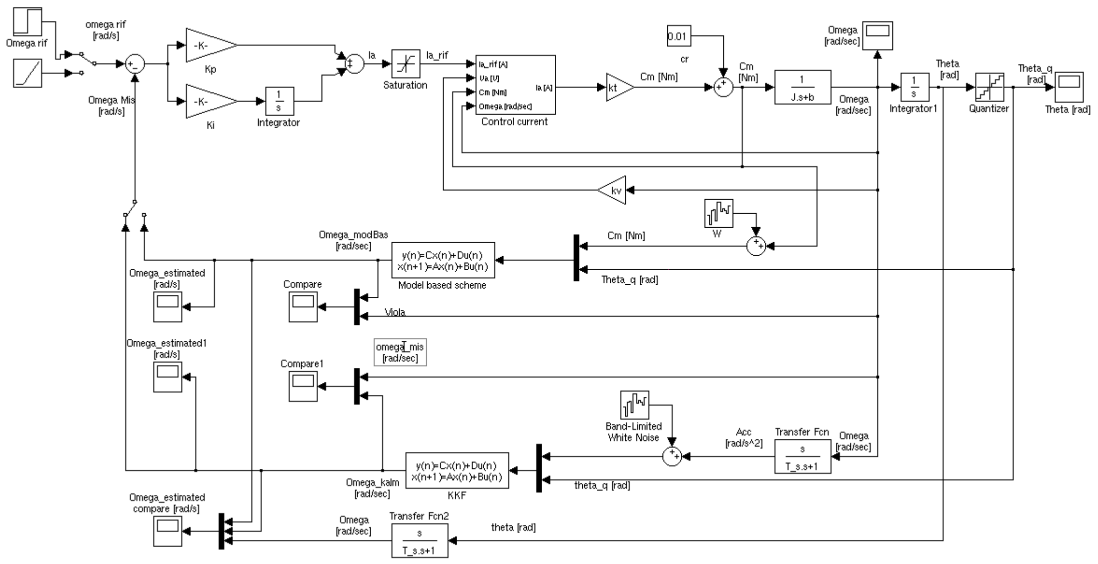


Figure 6.19: Full Simulink scheme with current control and speed. Velocity feedback from the model-base and KKF schemes.

Figure 6.20 shows the model of the current controller in detail.
Figure 6.21 shows Bode's diagram of the transfer function of the open-loop current controller. For the design of the two controllers I used the notes in [20]. I studied how to implement the errors of torque, and quantization so that they can implement the model as described in [1]. In fact, the model works perfectly without these errors, as expected, but needs the addition. This step is very difficult and must be done in the best way to respect as much as possible the real motor model.
In the future, a more accurate implementation of the model will be to consider in detail the motor with PWM control.

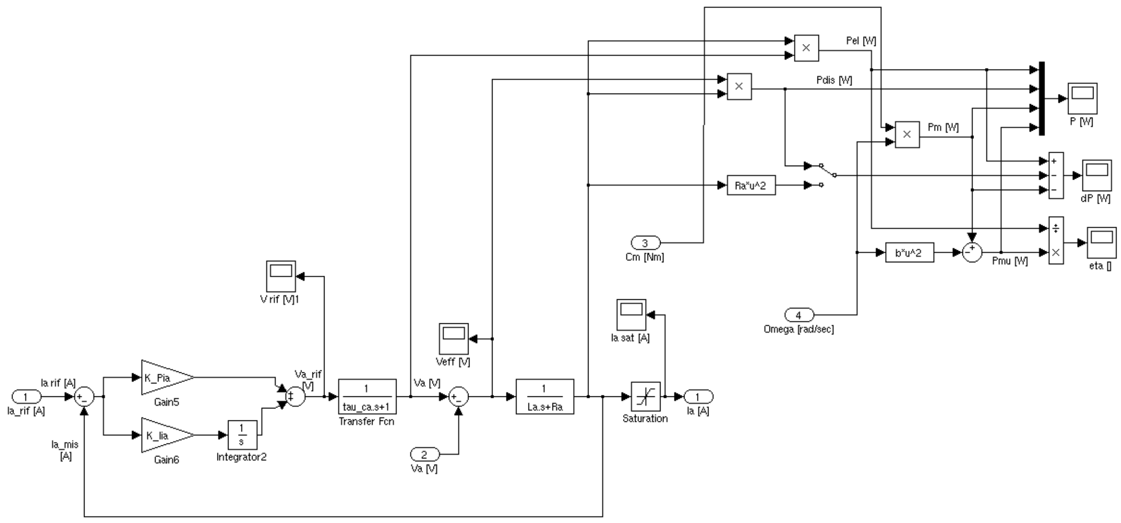


Figure 6.20: In detail: PI controller with current feedback.

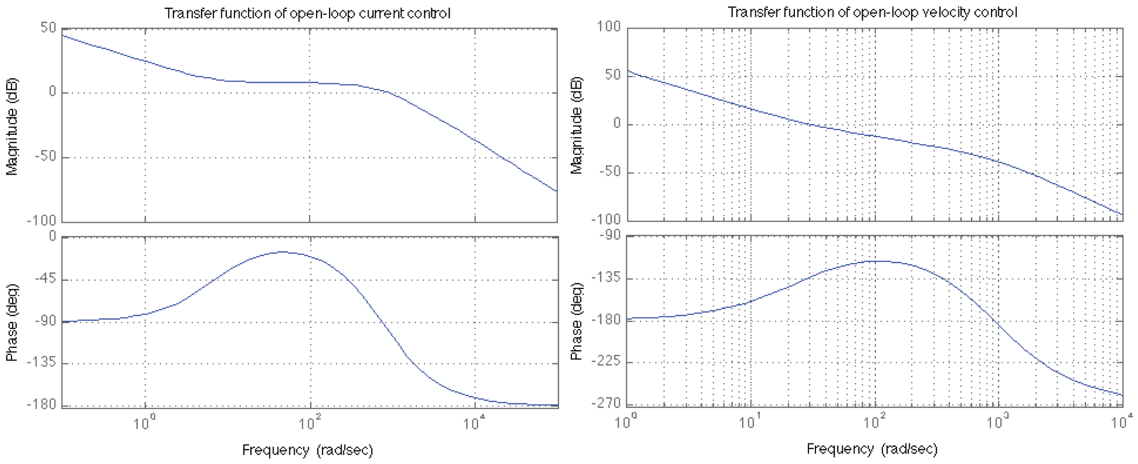


Figure 6.21: Bode's diagram of the transfer function of the open-loop current and velocity controller.

6.6 Week 4

This week I finished the implementation of the scheme in Simulink.
In the model-based scheme I added a filter in the disturbance observer in order to estimate the perturbation term.
The result obtained with the simulation appears similar to the result obteined in [1].

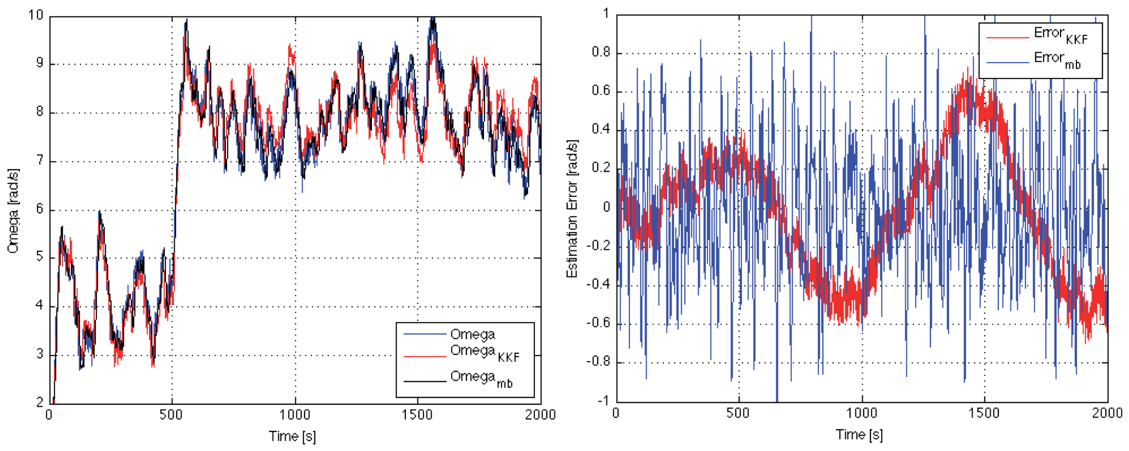


Figure 6.22: Velocity tracking with estimated velocities. Velocity feedback from the KKF. Left: velocity profiles. Right: Estimation error.

For Example, Figure 6.22 shows the simulation of the velocity estimated using velocity feedback from the Kalman Filter. Other simulations have been done also with the base-model feedback.

This week, after a meeting with my coordinator, we also decided to use the DSP card previously installed and I had to read the manuals in [30], [13] and [31].

I started to connect the encoder to the DSP card and review the possible differential line receiver. This in order to make a simple circuit to eliminate the measurement error of the encoder output.

For future work, I studied in detail the operation of synchronous motors with permanent magnets, isotropic and anisotropic, and their possible drives.

6.7 Week 5

This week I have connected the encoder in the DSP card. I noted that the DSP board doesn't need a filter upstream for the signal because it is already implemented in a noise filter in the card.
In fact, the encoder inputs are designed for incremental position sensors with differential outputs. The differential output lines of the sensor for the leading phase must be connected to the encoder inputs Phi0 and /Phi0, respectively. The same holds for the lagging phase encoder inputs Phi90, /Phi90 and for the index inputs.
The encoder's 24 bit counter value is scaled to ± 1.0.
To calculate the actual velocity use the following equation:

$$velocity = \frac{count \cdot 2\pi \cdot 1000}{4 \cdot PPR}, \left[\frac{rad}{\text{sec}}\right] \tag{6.6}$$

The speed is related to the precision of the encoder with the parameter PPR (Pulse Per Revolution), in our system equal to 4096.
The multiplication by 1000 is needed to estimate the speed per seconds, because the sampling time in the system is set at $T_s = 0.001$.
The division by 4 is needed, because the encoder interface uses fourfold multiplication for enhanced resolution, i.e. each encoder line produces 4 counts in the position counter.
Figure 6.23 shows the hardware design of the DSP PC board used in the system.

Unfortunately, the low signal from the encoder index doesn't work.
For this reason it was necessary to connect the signal index to an inverter chip so I can get the signal and connect it to the DSP card. To do this I used the MM74HC04N chip, Figure 6.24.
I studied how to design the speed controller and how to write in C language.
As a first step I considered the current proportional to the torque of the motor, so at the moment it only requires the design of speed control.

Monday next week I and Moss are going to connect the current loop in the driver.

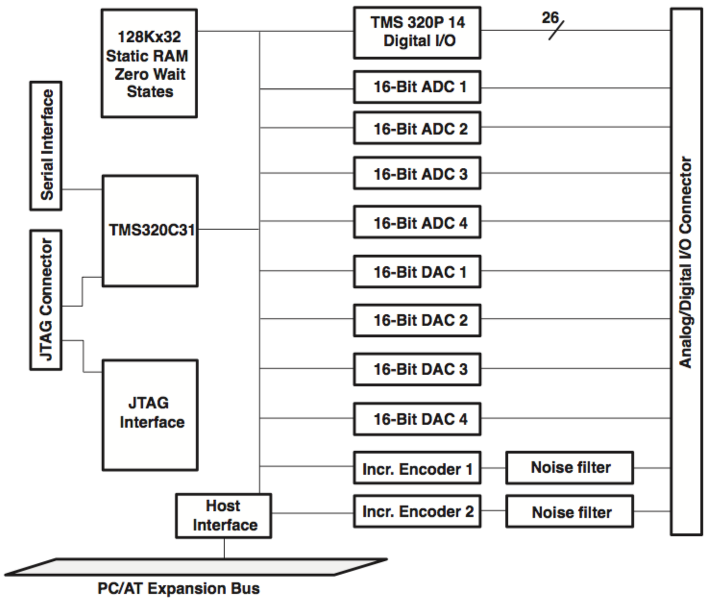


Figure 6.23: Hardware design of the DSP PC board used in the system.

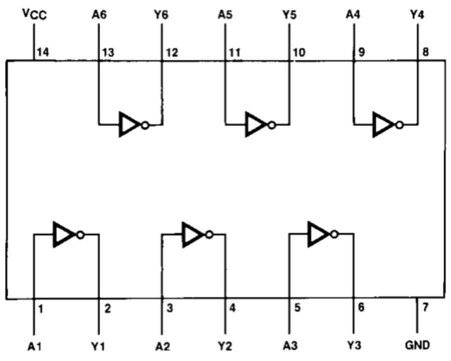


Figure 6.24: Inverter Chip.

6.8 Week 6

This week Moos has installed the current loop in the position indicated by the technicians of Moog. Figure 6.25 shows a diagram of the DS2100 driver and the two points where the input voltage is installed.

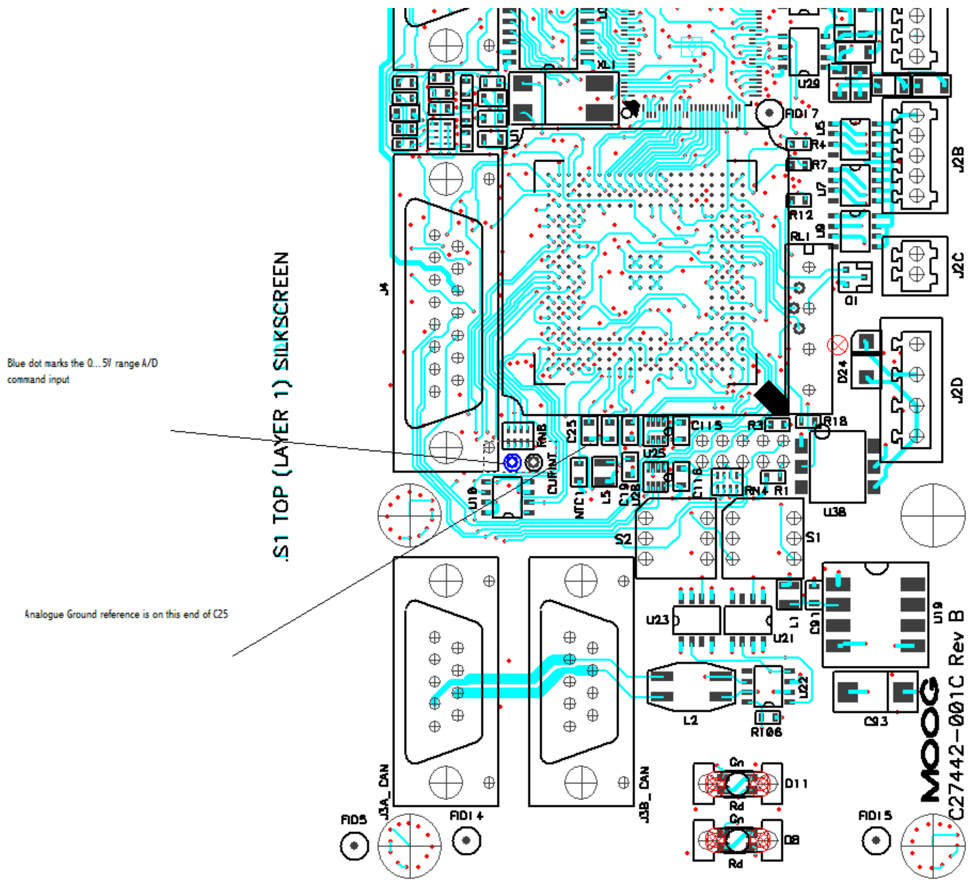

Blue dot marks the 0...5V range A/D command input
Analogue Ground reference is on this end of C25
S1 TOP (LAYER 1) SILKSCREEN
MOOG
C27442-001C Rev B
J2B
J2C
J2D
J3A_CAN
J3B_CAN
U19
U38
S1
S2
C25
FID5
FID14
FID15
D11
D8

Figure 6.25: DS2100, digital card layout.

At first the current-loop didn't work. Thanks to information from the MOOG engineers I was able to use the GUI in a command reference and analog torque mode.
To do this I had to change the parameter "modreq" making it equal to 8209, this means torque mode and use of the ADC command.
Particular attention is paid to the fact that the values of input voltage vary over a range of 0-4.85V and notes the calculations in the software assumes is biased at 2.44V.

Parameter in the database tab of the GUI the A/D input value can be viewed on adccmd_g parameter.
After some tests on the system it was found that the voltage and current supplied to the motor is equal:

- 0 V = -32704 adccmd_g increments (maps to +imax amps command to the current loop);
- 2.425 V = 0 adccmd_g increments (maps to 0 amps command to the current loop);
- 4.85 V = +32767 adccmd_g increments (maps to -imax amps command to the current loop).

So you need to supply a 2.425V bias on the input to get the command 0A current condition.

Another thing has been done this week: I have written the speed control in language C using the current-loop output from the DSP card. Before using this code on the system, there is a need to estimate/approximate the parameter of viscous damping and load torque in the system.
This is necessary for accurate calculation of the current feedback from the motor, in order to achieve reality as much as possible.
At the moment, with tests using the GUI and the current loop was extracted the static friction coefficient, which was equal to 0.39 Nm.
Next week I will test the motor to find a good ratio with which to estimate the load torque and the viscous damping.

6.9 Week 7

This week I calculated the coefficients of stiction and viscous damping level of the system.
To estimate these parameters the motor was controlled by a sine wave signal with amplitude of 50mV and frequency of 200mHz.
Figure 6.26 on the left, shows the voltage and corresponding current, in input to the system.

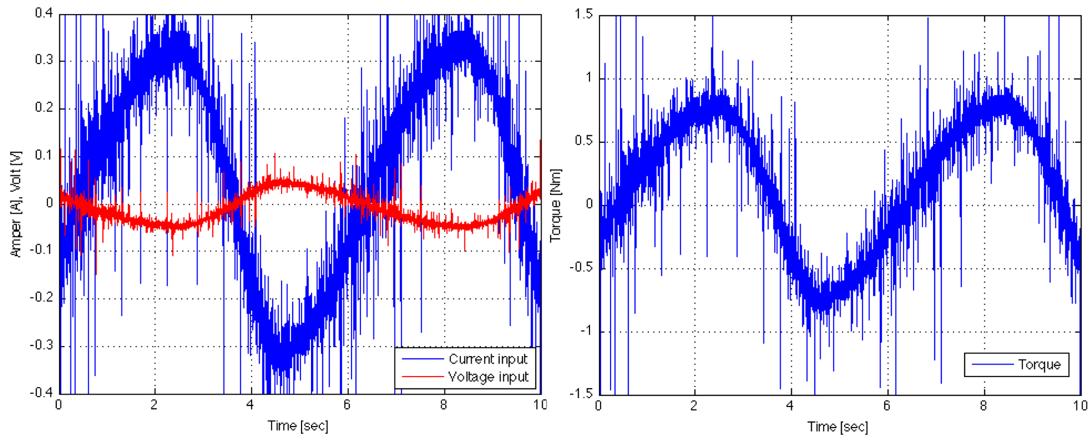


Figure 6.26: Left: Voltage and current in input. Right: Torque in input.

This correspondence is given by the written report in Chapter [6.8] and from some tests conducted using the GUI and the current-loop.
The relationship is given by the following equation:

$$\begin{aligned} current_{input} &= \frac{volt_{input} \cdot 17}{2.425}; \\ volt_{input} &= 2.425 - volt_{reference}. \end{aligned} \tag{6.7}$$

In the equation (6.7) $volt_{input}$ and $current_{input}$ are signals in input to the system, while the $volt_{reference}$ is the input signal present in the current-loop to the driver. Then, $volt_{reference}$ must be converted using the relationship in Chapter [6.8].
Knowing the current it is possible to calculate the torque applied to the motor using the following equation for permanent magnet synchronous motors:

$$m = \frac{3}{2} p \Lambda_{mg} i_q \tag{6.8}$$

where Λ_{mg} represents the maximum flux penetrating each phase due to the permanent magnet and p represents the number of motor poles. The value of $p\Lambda_{mg}$ was obtained by performing several tests on the system, comparing the current and torque values read by the GUI. The value is:

$$p\Lambda_{mg} = 1.58 \tag{6.9}$$

At this point it was possible to calculate the corresponding torque as shown in Figure 6.26 right. Figure 6.27 shows the acceleration and velocity measured in the system.

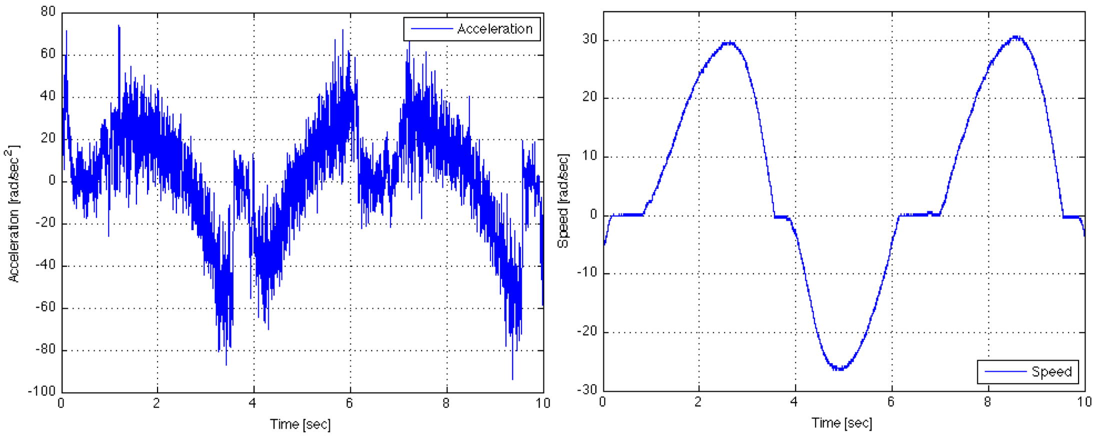


Figure 6.27: Left: Acceleration measured. Right: Velocity measured.

In this way, it was possible to estimate the value of the torque load; result is m_L = 0.3 [Nm]. Using the formula that represents the mechanical load:

$$m = m_L + B\omega_m + J\frac{d\omega_m}{dt} \tag{6.10}$$

it was possible to obtain an approximation of viscous damping (parameter B), which amounted to about B = 0.015 [Nm(s/rad)].

Knowing an approximation of the parameters that govern the system, I was able to test the PI controller. The PI controller designed by means of simulation shows a good response to steps of various amplitude. Figure 6.28 shows the response of the controller in two different steps.

If necessary, I can change the response of the controller by slightly varying the

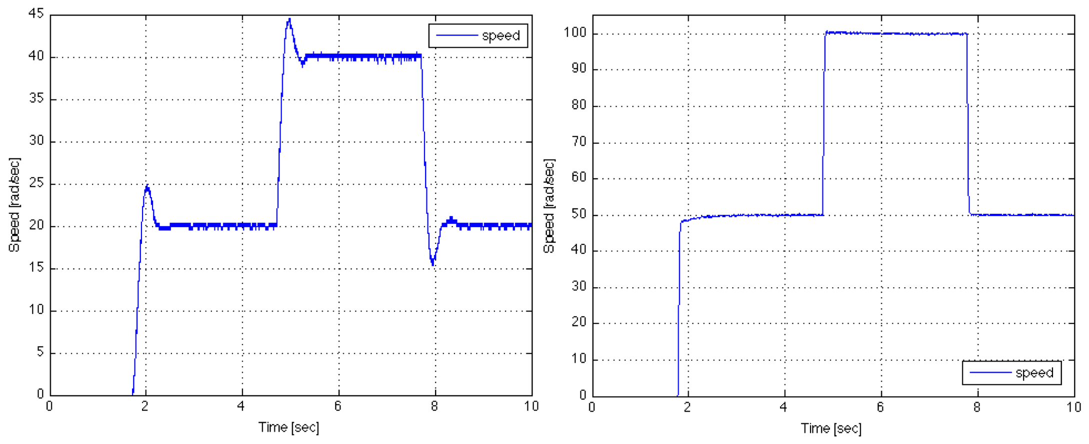


Figure 6.28: Response of the PI controller. Left: amplitude steps of 20 rad/sec. Right: amplitude steps of 50 rad/sec.

parameters K_p and K_i, in order to obtain a lower rise time and overshoot.

Next week I will finish writing the Kalman filter in C language in order to test it in the real system.

6.10 Week 8

This week I finished writing the discrete model of Model-Base and Kalman in C language.
I tested the two models in the real system using as feedback the motor speed in order to test the two models only if they pursued the real signal. In first time, the Model-Base worked quite well but the Kalman's model presented oscillations in the output signal. For this reason it was necessary to review the simulated model in order to find errors in the design of the two models. After a thorough verification of the matlab code, I found some mistakes in the description of noise in the model.

The curious fact is that in the simulation the errors were not noticed using Simulink, but the first time I noticed it in the real system. We understand the importance of designing a system simulation as accurate as possible and preferably without any errors. After correction of the errors found, the model-based model seems to work well but there are still some imperfections in the Kalman model.

Figure 6.29 shows the speed estimate from the Model-Base using steps of varying amplitude. As we can see the estimated speed follows the real speed signal well.
Figure 6.30 shows the speed estimate from the Kalman model using steps of varying amplitude. As we can see the real speed is not followed perfectly by the estimated speed, presenting fluctuations whenever the step change of amplitude.

Although I have long sought the error in the design I haven't found it yet; when I do the work of next week will improve the Kalman filter. This will make it possible to test the two filters using the feedback signal into the system by themselves.

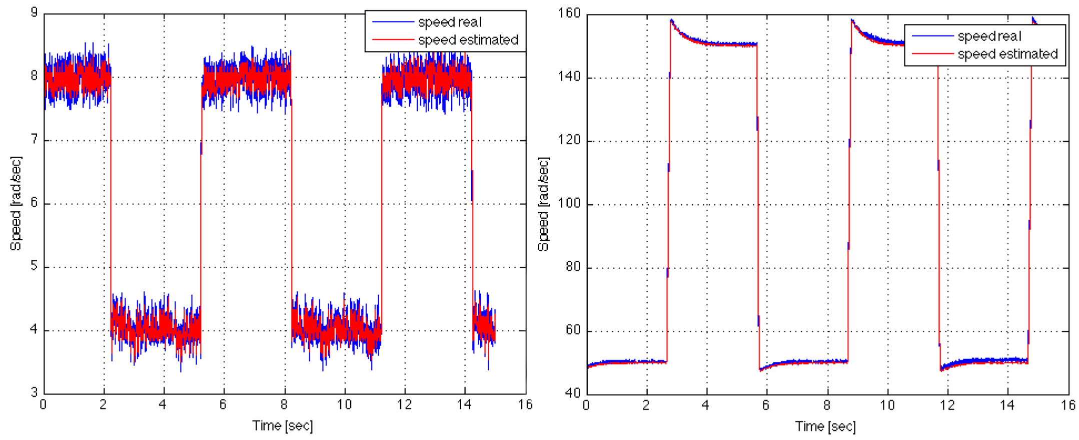


Figure 6.29: Response Model-Base using actual speed feedback. Left: amplitude steps of 4 rad/sec. Right: amplitude steps of 100 rad/sec.

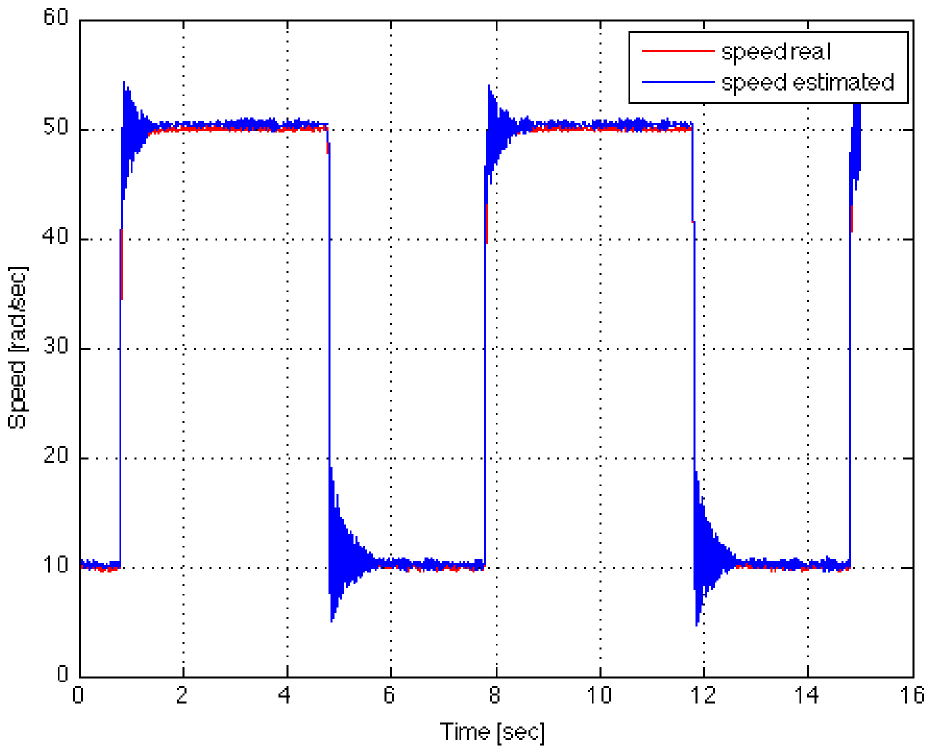


Figure 6.30: Response Kalman model using actual speed feedback. Amplitude steps of 40 rad/sec.

6.11 Week 9

This week I have implemented the two codes in order to use signals from the two filters as a feedback to the controller. The first code uses the signal feedback from the model-Base and the Kalman filter is used to compare the two estimated signals. The second code uses the signal feedback from the Kalman model and the model-base is used to compare the two estimated signals.

Unfortunately, a signal from an acceleration sensor was missing on the DSP, so I have found the cause of the failure. I found out that a sensor wasn't working and had to replace it. This took time, in fact to change the sensor I had to do precision welding because the wires are very thin and require attention in the welding. Moreover, I had to recalculate the sensitivity of the two sensors, which is different from the previous sensitivity.

With regard to the wide fluctuations in the signal estimated by the Kalman filter, these were due to the use of the acceleration signal filtered by a low pass filter implemented digitally in the code. Therefore, it was possible to use the speed feedback from the two different models and compare them. Figure 6.31 shows the speed as feedback signals using the two different models.

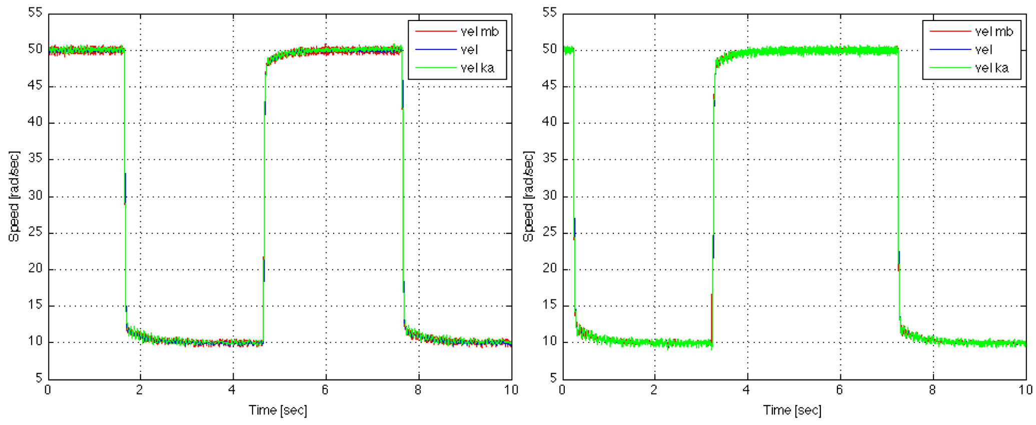


Figure 6.31: Velocity tracking with estimated velocities. Left: velocity profiles using velocity feedback from the model-based scheme. Right: velocity profiles using velocity feedback from the KKF.

To better understand, Figure 6.32 shows an enlargement of the two responses. Figure 6.33 also shows the speed estimation error of the two models. As you can see from the signals, the model-based tracks the signal better in both cases, if the speed feedback is by itself or the speed feedback is by the Kalman model. In fact the speed signal predicted by the model Kalman presents fluctuations that cause a large error in tracking. This behaviour is completely unexpected and it doesn't comply with what was previously done by simulation. I have tried in various ways to solve the problem but I haven't found a solution yet. I think that it has caused delay in the acceleration signal, which perhaps might cause these errors.

Next week it is important to find the cause of these errors in order to obtain the desired results.

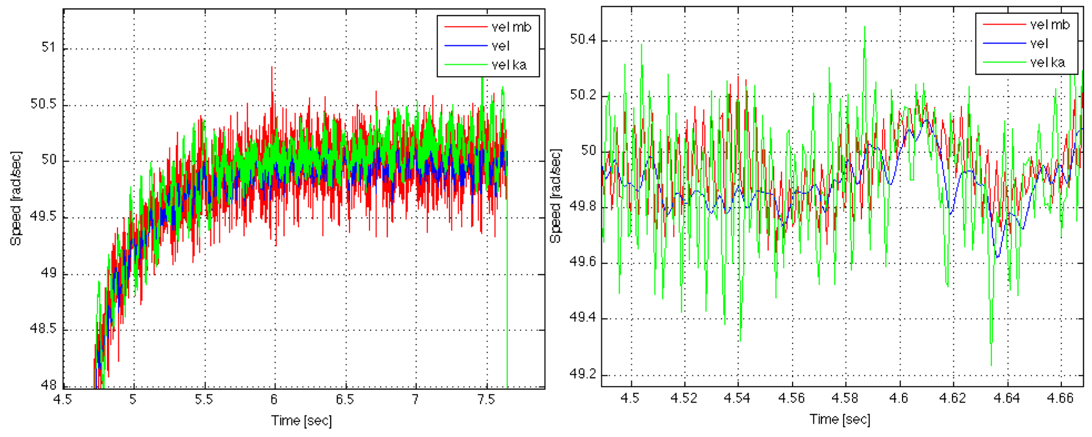


Figure 6.32: Velocity tracking with estimated velocities. Left: velocity profiles using velocity feedback from the model-based scheme. Right: velocity profiles using velocity feedback from the KKF.

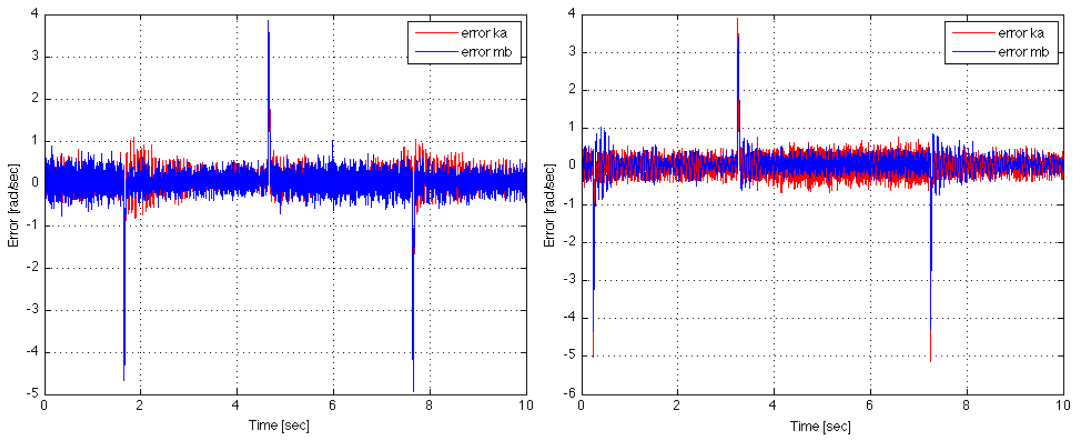


Figure 6.33: Velocity tracking with estimated velocities. Left: estimation errors using velocity feedback from the model-based scheme. Right: estimation errors using velocity feedback from the KKF.

6.12 Week 10

This week I have implemented the code by inserting the filter for the disturbance observer. I corrected some errors in the code regarding the input signals in the two filters.

I also noticed an improvement in the response of the Kalman filter using a smaller variance due to the quantization error of the encoder. This implies that the Kalman filter has a smaller estimation gain and it means slower estimator dynamics. This makes the Kalman filter less sensitive to the quantization effect. As described in [1], the slower estimator dynamics doesn't necessarily mean larger phase delay. This means the estimation is less dependent on the position signal which is corrupted by the quantization error and more dependent on the acceleration measurement. Therefore, the estimation can be much more accurate even with slower estimator dynamics if I can get a good acceleration measurement, which is the input to the system in the Kalman filter.

Disturbance $\hat{d}$ can be represented by u and y as

$$\hat{d}(s) = D(s)(Q(s)y(s) - u(s)) \tag{6.11}$$

In other words, Q(s) is nothing but a low pass filter with unity gain and D(s) is the inverse of the nominal plant. Note that the order of the low pass filter Q(s) is three and its parameters depend on the estimation gain of the filter, which means that the design of the disturbance observer is coupled with the design of the state estimator.

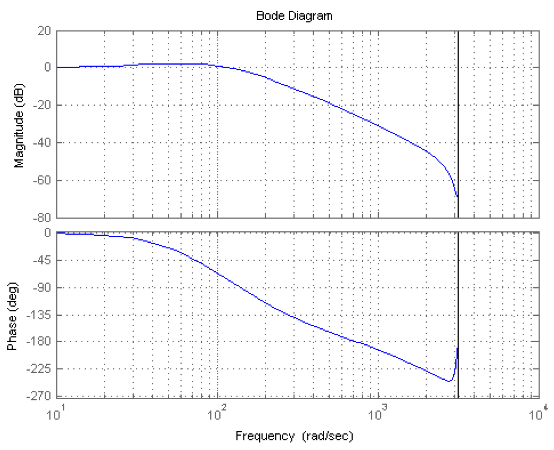
Therefore, in the model-based velocity estimator has implemented the following low-pass filter in the disturbance observer in the discrete time:

$$Q(z) = \frac{1.373 \cdot 10^{-2} z^{-1} - 7.176 \cdot 10^{-4} z^{-2} - 1.215 \cdot 10^{-2} z^{-3}}{1 - 2.715 z^{-1} + 2.456 z^{-2} - 7.408 \cdot 10^{-1} z^{-3}} \tag{6.12}$$

I also used the following high-pass filter:

$$D(z) = \frac{6.27 - 18.15 z^{-1} + 17.51 z^{-2} - 5.625 z^{-3}}{1 - 2.715 z^{-1} + 2.456 z^{-2} - 7.408 \cdot 10^{-1} z^{-3}} \tag{6.13}$$

Figure 6.34 shows the Bode plots of the two filters.



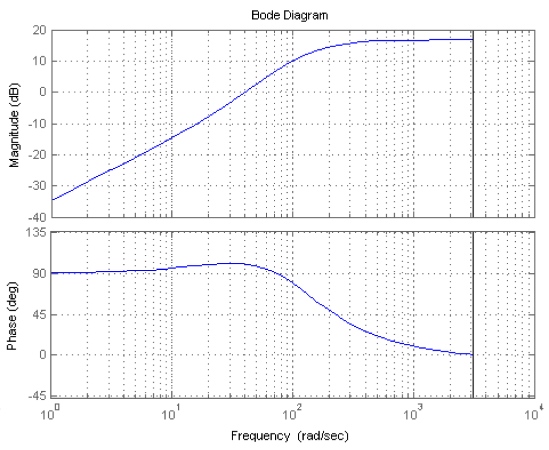


Figure 6.34: Left: Bode's diagram of the transfer function G(z). Right: Bode's diagram of the transfer function D(z).

The two filters have been implemented in the code in C.

Unfortunately, they were not tried in the system because there was a malfunction in the acceleration signal, similar to that of last week. Initially I thought of a break of another accelerometer but when this was replaced the problem persisted. I discovered that there was a short circuit in the wiring of the Mercury Slip Ring inside the shaft. This problem solved, a new one appeared; a support bearing shaft was broken, causing considerable friction.

Next week I will have to change the broken bearing and finally I will be able to test the final code in the real system and acquire the necessary data.

6.13 Week 11

This week I have fixed the broken bearing and made several tests on the system. Firstly, I have conducted some tests to find the best value of the error variance (W) using the model-based, precisely using a quantized signal of the position with PPR = 4096 and PPR = 256. The same operation was done to find the best value of the variance of the noise of the acceleration (W_a) using two different quantization encoders.
The results are shown in Figures 6.35,6.36,6.37,6.38. It was discovered that using the encoder signal with high resolution, the best value of noise is W $= 7.5\times10^{-4}$ and $W_a = 10$, for the model-base and Kalman filter respectively. These values have a lower maximum error of estimation of the signal. When using an encoder with low resolution the results are very different. The best values are W $= 1\times10^{-2}$ and $W_a = 500$, greater than those used with a high resolution encoder.

A curious thing was that by including too many operations in the C code, such as signal filtering and signal estimation, the output signal was saturated and had very large swing. After some research it was found that the processor was in overflow.

Next week I am going to test the new filters for the estimation error and I am going to perform other tests in order to collect the necessary data to compare the two models.

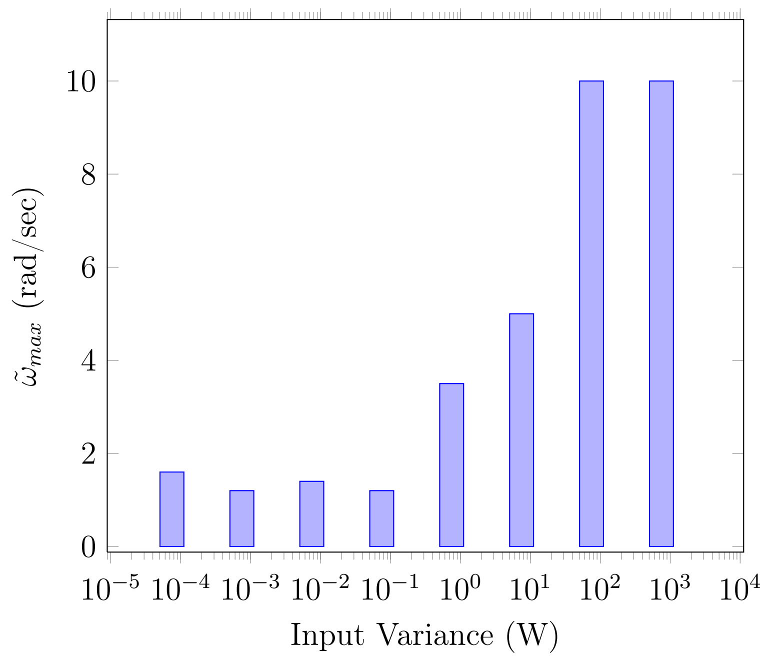


Figure 6.35: Velocity estimation error. Model-base estimator with high resolution encoder.

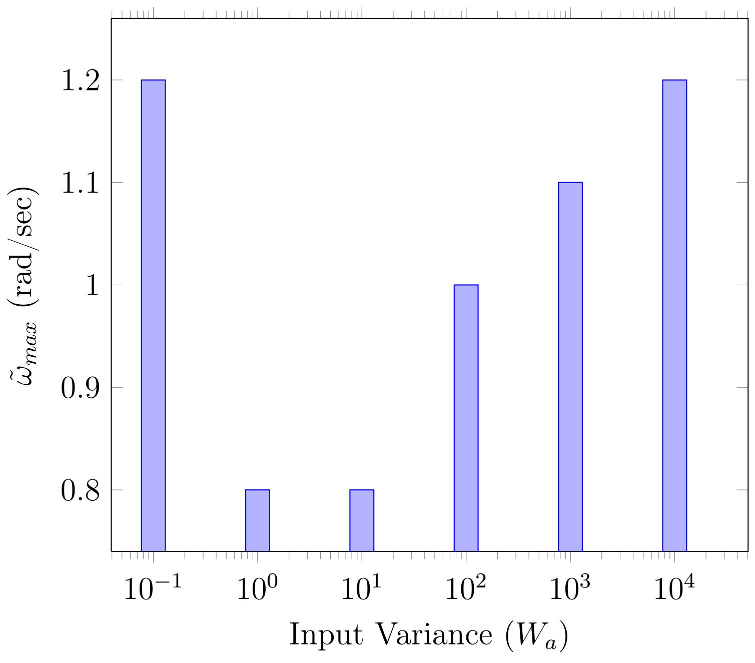


Figure 6.36: Velocity estimation error. Kalman estimator with high resolution encoder.

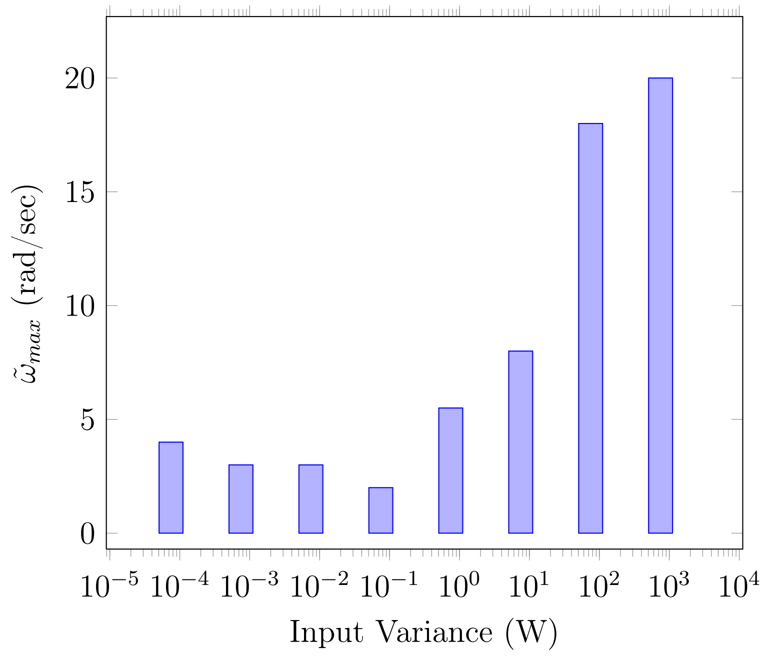


Figure 6.37: Velocity estimation error. Model-base estimator with low resolution encoder.

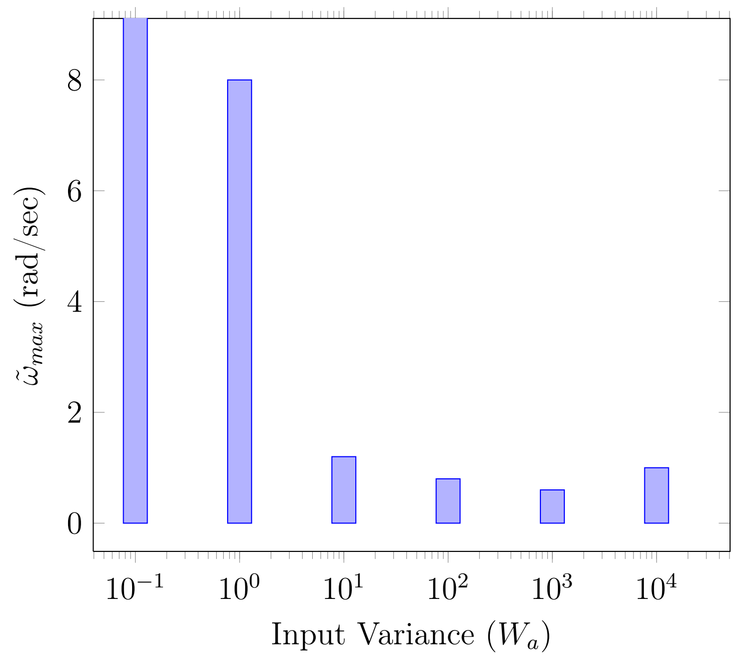


Figure 6.38: Velocity estimation error. Kalman estimator with low resolution encoder.

6.14 Week 12

This week I have found an error in the positioning of devices which included an erroneous offset in the output signal of the accelerometers. I also fixed a bug that was present in the code implemented in C language and therefore it was necessary to redo the tests conducted last week to find out the best values of the variances W and W_a.

Therefore, the diagrams and data presented last week have been changed. In fact, I have found a good results with regard to the use of KKF with values of Wa lower than those reported last week. Therefore, various tests were performed on the system with the use of two estimators in order to collect the necessary data.

The data and the various diagrams will be directly reported in the thesis in order to do not make unnecessary repetitions.

Appendix A

Code C

```c
#include "brtenv.h"
#include "math.h"

#define DT 100e-5
#define N 100
#define NZEROS 6  /* Cut-Off 1000Hz */
#define NPOLES 6
#define GAIN   3.198759421e+02

/* Parameter Declaration */
float ADC_in1=0;
float ADC_in2=0;
float ADC_in3=0;
float ADC_in4=0;

float encoder =0;
float ADC_1filt;
float ADC_2filt;
float Hoodwin = 0;
float accel=0;
float exec_time=0;
float analog_avg=0;
float average =0;
float average_filt =0;
float PPR = 4096;
float offset = 2.45;
float position = 0;
float position_old = 0;
float speed = 0;
float speed_ref = 0;
float speed_error = 0;
float speed_error_int = 0;
float acc_diff =0;
float acc_linear = 0;
float acc_angular = 0;
float current_loop = 0;
float current_error_old= 0;
float current_error = 0;
float current_error_shift = 0;
float current_PI = 0;
float torque = 0;
float Volt_out = 0;
float Volt_tmp = 0;
float counter = 0;
float output = 0;

/* Parameter System */
float B = 0.015; /* coefficient viscous damping [Nms] */
float J = 0.000209; /* rotor inertia with resolver [kgm^2] */
float sensitivity = 100.51282; /* sensitivity for 1 V in m/s^2 */
float R = 0.055; /* radius accelerometer  [m] */
float p_gamma =1.58; /* p*gamma_mg */
float m_load = 0.3;
/*Max value*/
float Vmax = 5;
float current_max = 2;

/* PI Velocity Control Parameter */
float Pprop = 0;
float Kp = 0.086;
float Ki  = 0.2;

/* Array's for Butter filter*/
```

```
static float Array[N-1];
static float xv[NZEROS+1], yv[NPOLES+1],xv2[NZEROS+1],
yv2[NPOLES+1];

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Digital Filter 6th Order LP Butter Filter (Flat Band
Characteristics) fc=250Hz */

static void filterloop()   {

 xv[0] = xv[1]; xv[1] = xv[2]; xv[2] = xv[3]; xv[3] = xv[4]; xv
     [4] =
xv[5]; xv[5] = xv[6];
       xv[6] = ADC_in1 / GAIN;
       yv[0] = yv[1]; yv[1] = yv[2]; yv[2] = yv[3]; yv[3] = yv
          [4];
yv[4] = yv[5]; yv[5] = yv[6];
       yv[6] =   (xv[0] + xv[6]) + 6 * (xv[1] + xv[5]) + 15 * (xv
          [2] + xv[4])
                    + 20 * xv[3]
                          + ( -0.0078390522 * yv[0]) + (
                             0.0852096278 * yv[1])
                    + ( -0.4080412916 * yv[2]) + (  1.1157139955
                       * yv[3])
                    + ( -1.8767603680 * yv[4]) + (  1.8916395224
                       * yv[5]);
       ADC_1filt = yv[6];
     }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
static void filterloop2()   {

xv2[0] = xv2[1]; xv2[1] = xv2[2]; xv2[2] = xv2[3]; xv2[3] = xv2
    [4];
xv2[4] = xv2[5]; xv2[5] = xv2[6];
       xv2[6] = ADC_in2 / GAIN;
       yv2[0] = yv2[1]; yv2[1] = yv2[2]; yv2[2] = yv2[3]; yv2[3]
          =
yv2[4]; yv2[4] = yv2[5]; yv2[5] = yv2[6];
       yv2[6] =   (xv2[0] + xv2[6]) + 6 * (xv2[1] + xv2[5]) + 15
          *
(xv2[2] + xv2[4])
                    + 20 * xv2[3]
                    + ( -0.0078390522 * yv2[0]) + (  0.0852096278
                        * yv2[1])
                    + ( -0.4080412916 * yv2[2]) + (  1.1157139955
                        * yv2[3])
                    + ( -1.8767603680 * yv2[4]) + (  1.8916395224
                        * yv2[5]);
       ADC_2filt = yv2[6];
     }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

unsigned int   err_cnt;
/* error flag for CHKERRXX at last dual-port memory location
             */
int *error = (int *) (DP_MEM_BASE + DP_MEM_SIZE - 1);

/*------------------------------------------------------------*/
       isr_t0()
       {
               begin_isr_t0(*error);

                service_trace();
                count0 = count_timer(0);

                 ADC_in1 = ds1102_ad(1);
                 ADC_in2 = ds1102_ad(2);
/* 1 */
ds1102_ad_start();      /* starts ADC conversion */
                 ADC_in1 = ADC_in1 + ds1102_ad(1);
                 ADC_in2 = ADC_in2 + ds1102_ad(2);

/* 2 */
ds1102_ad_start();      /* starts ADC conversion */
```

```
                      ADC_in1 = ADC_in1 + ds1102_ad(1);
                      ADC_in2 = ADC_in2 + ds1102_ad(2);

ds1102_ad_start();
                      ADC_in1 = 10*ADC_in1/3;
                      ADC_in2 = 10*ADC_in2/3;

analog_avg = 10*ds1102_ad(3);
Hoodwin      = 10*ds1102_ad(4);

filterloop();
filterloop2();

average_filt = (ADC_1filt + ADC_2filt)/2;
average = (ADC_in1 + ADC_in2)/2;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Regolation Steps input*/

counter = counter + 1;
if(counter == 3000){
      speed_rif = 50;}
 else if(counter == 6000) {
       speed_rif = 200;
       counter = 0;
}
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Counts Encoder */
encoder = (ds1102_inc(1)*8388608);

/* Position */
position_old = position;

position = -(encoder*2*3.1415926)/(4*PPR);

/* Velocity [rad/sec] */
speed = -(encoder*2*3.1415926*1000)/(4*PPR);

```

```
/*  Clearing the Counter */
ds1102_inc_clear_counter(1);

if(position >position_old + 10) position = position_old;
       else if(position < (position_old-10)) position =
          position_old;
       else position = position ;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Calculating Acceleration  */
/* Acceleration difference */

acc_diff = average - offset;

/* Calculation Linear Acceleration */
/* sensibility  1V is 100,51282 m/s^2  */

acc_linear = acc_diff*sensitivity;

/* Calculation Angular Acceleration */
acc_angular = acc_linear/R;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Torque and Current  Calculation
m = m_load + B*speed + J*acceleration
m = (3/2)*p*gamma_mg*i_q */

torque = m_load + B*speed + J*acc_angular;

current_loop = (2/3)*(torque/p_gamma);

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Control Speed */
speed_error = speed_ref - speed;

Pprop = (speed_error)*Kp;

speed_error_int = speed_error_int + (speed_error)*DT*Ki;
```

```

current_PI = Pprop + speed_error_int;  /* PI controller output*/

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Saturation to Limit the Current Reference to 15A Peak Current
    */

if(current_PI>current_max)current_PI=current_max;
       else if(current_PI < -current_max) current_PI = -
          current_max;
       else current_PI = current_PI;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* current Error */

current_error_old = current_error;

current_error = current_PI - current_loop;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Second Saturation for Safety */
if(current_error>current_max) current_error =current_max;
       else if(current_error< -current_max) current_error = -
          current_max;
       else current_error = current_error;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Piggy output */
Volt_tmp = (current_error*2.425)/17;

Volt_out = 2.425 - Volt_tmp;

/* Third Saturation for Safety */
if(Volt_out >3.5) Volt_out =3.5;
       else if(Volt_out < 1.5) Volt_out  = 1.5;
       else Volt_out  = Volt_out ;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
ds1102_da(1,ADC_in1/10);
ds1102_da(2,ADC_in2/10);
ds1102_da(3,speed/1000);
ds1102_da(4,Volt_out/10);

                 exec_time = time_elapsed(0, count0);

     end_isr_t0();
}

/*--------------------------------------------------------------*/
/* Main Function */

main()
{
 int i=0;

 init(); /* Initialize Hardware System */
 *error = NO_ERROR; /* Initialize Error Flag */

  start_isr_t0(DT);

err_cnt = 0;
CHECKERR:
 while (*error == NO_ERROR); /* Background Process */
 *error = NO_ERROR;

 init(); /* Initialize Hardware System */
 start_isr_t0(DT);
 err_cnt = err_cnt + 1;
 goto CHECKERR;
}
```

Listing A.1: PI.c

```
#include "brtenv.h"
#include "math.h"

#define DT 100e-5
```

```
#define N 100
#define NZEROS 6  /* Cut-Off 1000Hz */
#define NPOLES 6
#define GAIN   3.198759421e+02

/* Parameter Declaration */
float ADC_in1=0;
float ADC_in2=0;
float ADC_in3=0;
float ADC_in4=0;

float encoder =0;
float average =0;
float PPR = 4096;
float PPR1 = 256;
float offset = 2.508;
float position = 0;
float position_old = 0;
float position_output = 0;
float position_output_old = 0;
float position_q = 0;
float position_quan = 0;
float position_old_quan = 0;
float speed = 0;
float speed_ref = 0;
float speed_error = 0;
float speed_error_int = 0;
float speed_filt;
float speed_estimated_ka = 0;
float speed_estimated_mb = 0;
float acc_diff =0;
float acc_linear = 0;
float acc_angular = 0;
float acc_angular_old = 0;
float current_loop = 0;
float current_error = 0;
float current_PI = 0;
float torque = 0;
float torque_input = 0;
float torque_old_input = 0;
float torque_old = 0;
float Volt_out = 0;
float Volt_tmp = 0;
float counter = 0;

/* Parameter System */
float B = 0.015; /* coefficient viscous damping [Nms] */
float J = 0.000209; /* Rotor inertia with resolver [kgm^2] */
float sensitivity = 100.51282;/ * sensitivity for 1 V in m/s^2 */
float R = 0.055; /* radius accelerometer  [m] */
float p_gamma =1.58;    /* p*gamma_mg */
float m_load = 0.3;

/*Max value*/
float Vmax = 5;
float current_max = 2;

/* PI control velocity value */
float Pprop = 0;
float Kp = 0.086;
float Ki  = 0.2;  /* Original value = 0.2152 */

/* Array's for Butter filter*/
static float Array[N-1];
static float xv[NZEROS+1], yv[NPOLES+1],xv2[4], yv2[4],xv3[4],
    yv3[4];

/* Parameter KKF */
double  x0_ka[2] = {0,0};
double  u_ka[2]={0,0};

/* Parameter Model-Base scheme */
double  x0_mb[2] = {0,0};
double  u_mb[2]={0,0};

float Q_filt = 0;
float pos_filt = 0;
float estimation_error=0;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
static void filterloop()   {

xv[0] = xv[1]; xv[1] = xv[2]; xv[2] = xv[3]; xv[3] = xv[4];
xv[4] = xv[5]; xv[5] = xv[6];
       xv[6] = speed / GAIN;
       yv[0] = yv[1]; yv[1] = yv[2]; yv[2] = yv[3]; yv[3] =
yv[4]; yv[4] = yv[5]; yv[5] = yv[6];
       yv[6] =   (xv[0] + xv[6]) + 6 * (xv[1] + xv[5]) + 15 *
(xv[2] + xv[4])
                    + 20 * xv[3]
                    + ( -0.0078390522 * yv[0]) + (  0.0852096278
                       * yv[1])
                    + ( -0.4080412916 * yv[2]) + (  1.1157139955
                       * yv[3])
                    + ( -1.8767603680 * yv[4]) + (  1.8916395224
                       * yv[5]);
       speed_filt = yv[6];
     }
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/*%%%%%%%%%%%%%%%%%% Disturbance Observer
   %%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Low-pass Filter Q(z) */

static void filterloop2()   {

xv2[0] = xv2[1]; xv2[1] = xv2[2]; xv2[2] = xv2[3];
xv2[3] = torque_input;
       yv2[0] = yv2[1]; yv2[1] = yv2[2]; yv2[2] = yv2[3];
       yv2[3] =   -0.0121*xv2[0] -0.0007158*xv2[1] +0.01373*xv2
          [2] +0*xv2[3] +0.7408*yv2[0] - 2.4562*yv2[1] +2.7145*
          yv2[2];
       Q_filt = yv2[3];
     }
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
/*  High-pass Filter D(z)  */

static void filterloop3()   {

xv3[0] = xv3[1]; xv3[1] = xv3[2]; xv3[2] = xv3[3];
xv3[3] = position_output_old;
       yv3[0] = yv3[1]; yv3[1] = yv3[2]; yv3[2] = yv3[3];
       yv3[3] =  6.27*xv3[3] -18.1497*xv3[2] + 17.5051*xv3[1]
          -5.6254*xv3[0] +0.7408*yv3[0] - 2.4562*yv3[1] +2.7145*
          yv3[2];
       pos_filt = yv3[3];
}
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Noice variance accelerometer Wa = 5 [rad/sec^2] */
/* n = #states, m = #outputs, r = #inputs */

enum {n_Model_kalman = 2, m_Model_kalman = 1, r_Model_kalman =
   2};

void Initialize_Model_kalman(const double* x0_ka);
void Update_Model_kalman(const double* u_ka);
const double *Output_Model_kalman();
const double *State_Model_kalman();

static const double a_ka[n_Model_kalman*n_Model_kalman] =
{
      9.043995286e-01,   9.043995286e-04,
     -4.802151624e+00,   9.951978484e-01
};

static const double b_ka[n_Model_kalman*r_Model_kalman] =
{
     4.521997643e-07,   9.560047136e-02,
     9.975989242e-04,   4.802151624e+00
};

static const double c_ka[m_Model_kalman*n_Model_kalman] =
{
     0.000000000e+000,  1.000000000e+000
};

static const double d_ka[m_Model_kalman*r_Model_kalman] =
{
     0.000000000e+000,  0.000000000e+000
};

static double x_ka[n_Model_kalman], y_ka[m_Model_kalman];

void Initialize_Model_kalman(const double* x0_ka)
{
    int i;

    /* Initialize x */
    for (i=0; i<n_Model_kalman; i++)
        x_ka[i] = x0_ka[i];
}

void Update_Model_kalman(const double* u_ka)
{
    int i, j;
    double x_next_ka[n_Model_kalman];

    /* Evaluate x_next = A*x + B*u */
    for (i=0; i<n_Model_kalman; i++)
    {
        x_next_ka[i] = 0;
        for (j=0; j<n_Model_kalman; j++)
            x_next_ka[i] += a_ka[i*n_Model_kalman+j]*x_ka[j];

        for (j=0; j<r_Model_kalman; j++)
            x_next_ka[i] += b_ka[i*r_Model_kalman+j]*u_ka[j];
    }

    /* Evaluate y = C*x + D*u */
    for (i=0; i<m_Model_kalman; i++)
    {
        y_ka[i] = 0;
        for (j=0; j<n_Model_kalman; j++)
            y_ka[i] += c_ka[i*n_Model_kalman+j]*x_ka[j];

        for (j=0; j<r_Model_kalman; j++)
            y_ka[i] += d_ka[i*r_Model_kalman+j]*u_ka[j];
    }

    /* Update x to its next value */
    for (i=0; i<n_Model_kalman; i++)
        x_ka[i] = x_next_ka[i];
}

const double *Output_Model_kalman()
{
    return y_ka;
}

const double *State_Model_kalman()
{
    return x_ka;
}

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Model Base Filter*/
/* stiction level 0.3 Nm*/
/* n = #states, m = #outputs, r = #inputs */

enum {n_Model_Base = 2, m_Model_Base = 1, r_Model_Base = 2};

void Initialize_Model_Base(const double* x0_mb);
void Update_Model_Base(const double* u_mb);
const double *Output_Model_Base();
const double *State_Model_Base();

static const double a_mb[n_Model_Base*n_Model_Base] =
{
     2.800397348e-01,   2.702266228e-04,
    -4.435102419e+02,   5.027758108e-01
};

static const double b_mb[n_Model_Base*r_Model_Base] =
{
     6.542074669e-04,   7.199602652e-01,
     3.580929817e+00,   4.435102419e+02
};

static const double c_mb[m_Model_Base*n_Model_Base] =
{
     0.000000000e+000,  1.000000000e+000
};

static const double d_mb[m_Model_Base*r_Model_Base] =
{
     0.000000000e+000,  0.000000000e+000
};

static double x_mb[n_Model_Base], y_mb[m_Model_Base];

void Initialize_Model_Base(const double* x0_mb)
{
    int i;

    /* Initialize x */
    for (i=0; i<n_Model_Base; i++)
        x_mb[i] = x0_mb[i];
}

void Update_Model_Base(const double* u_mb)
{
    int i, j;
    double x_next_mb[n_Model_Base];

    /* Evaluate x_next = A*x + B*u */
    for (i=0; i<n_Model_Base; i++)
    {
```

```c
        x_next_mb[i] = 0;
        for (j=0; j<n_Model_Base; j++)
            x_next_mb[i] += a_mb[i*n_Model_Base+j]*x_mb[j];

        for (j=0; j<r_Model_Base; j++)
            x_next_mb[i] += b_mb[i*r_Model_Base+j]*u_mb[j];
    }

    /* Evaluate y = C*x + D*u */
    for (i=0; i<m_Model_Base; i++)
    {
        y_mb[i] = 0;
        for (j=0; j<n_Model_Base; j++)
            y_mb[i] += c_mb[i*n_Model_Base+j]*x_mb[j];

        for (j=0; j<r_Model_Base; j++)
            y_mb[i] += d_mb[i*r_Model_Base+j]*u_mb[j];
    }

    /* Update x to its next value */
    for (i=0; i<n_Model_Base; i++)
        x_mb[i] = x_next_mb[i];
}

const double *Output_Model_Base()
{
    return y_mb;
}

const double *State_Model_Base()
{
    return x_mb;
}
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
unsigned int   err_cnt;
/* error flag for CHKERRXX at last dual-port memory location
             */
int *error = (int *) (DP_MEM_BASE + DP_MEM_SIZE - 1);
```

```

/*-------------------------------------------------------------*/
       isr_t0()
       {
               begin_isr_t0(*error);

                service_trace();
                count0 = count_timer(0);

                 ADC_in1 = ds1102_ad(1);
                 ADC_in2 = ds1102_ad(2);
/* 1 */
ds1102_ad_start();      /* starts ADC conversion */
                 ADC_in1 = ADC_in1 + ds1102_ad(1);
                 ADC_in2 = ADC_in2 + ds1102_ad(2);

/* 2 */
ds1102_ad_start();      /* starts ADC conversion */
                 ADC_in1 = ADC_in1 + ds1102_ad(1);
                 ADC_in2 = ADC_in2 + ds1102_ad(2);
ds1102_ad_start();
               ADC_in1 = 10*ADC_in1/3;
               ADC_in2 = 10*ADC_in2/3;

average = (ADC_in1 + ADC_in2)/2;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* regulation steps input */

counter = counter + 1;
if(counter == 6000)  {
      speed_rif = 4;}
else if(counter == 9000) {
      speed_rif = 8;
      counter = 0;       }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Counts encoder */
```

```
encoder = -(ds1102_inc(1)*8388608);

position_old_quan = position_quan;

position_q = (int)((PPR1/(PPR))*encoder);

position_quan = (position_q/(PPR1*4))*2*3.1415926;

/* Position calculation */

position_old = position;

position = (encoder*2*3.1415926)/(4*PPR);

/* velocity in [rad/sec] */

 speed = (position - position_old)/DT;

/* Code to avoid encoder error */

if(position >position_old + 10) position = position_old;
       else if(position < (position_old-10)) position =
          position_old;
       else position = position ;

if(position_quan >position_old_quan + 10) position_quan =
   position_old_quan ;
       else if(position_quan  < (position_old_quan -10))
          position_quan  = position_old_quan ;
       else position_quan  = position_quan;

position_output_old = position_output;

/* position in input to the KKF and Model-based scheme */
/* position calculation */

position_output = position_quan;

```

```
/* filtering of the velocity */
filterloop();

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Calculating acceleration  */

/* Acceleration difference */
acc_diff = average  - offset;

/* calculation linear acceleration */
/* sensivity  1V is 100,51282 m/s^2  */
acc_linear = acc_diff*sensitivity;

/* calculation angular acceleration */
acc_angular_old = acc_angular;

acc_angular = acc_linear/R;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Torque and current  calculation
m = m_load + B*speed + J*acceleration
m = (3/2)*p*gamma_mg*i_q*/

torque_old = torque;

torque = m_load + B*speed_estimated_ka + J*acc_angular_old;

torque_old_input = torque_input;

torque_input  = B*speed_estimated_ka + J*acc_angular_old;

current_loop = (2/3)*(torque/p_gamma);

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

filterloop2();
filterloop3();

```

```
estimation_error = pos_filt - Q_filt;
/*
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    */
/* State estimation using Kalman Filter */
x0_ka[0]= 0;
x0_ka[1]= 0;

if(count_initialize == 0)Initialize_Model_kalman(x0_ka);

u_ka[0] =  acc_angular_old;
u_ka[1] =  position_output;

Update_Model_kalman(u_ka);
Output_Model_kalman();

speed_estimated_ka = y_ka[0];

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* State estimation using the Model-Base Filter */
x0_mb[0]=0;
x0_mb[1]=0;

if(count_initialize == 0)Initialize_Model_Base(x0_mb);
count_initialize += 1;

u_mb[0] = torque_input + estimation_error;
u_mb[1] = position_output;

Update_Model_Base(u_mb);
Output_Model_Base();
speed_estimated_mb = y_mb[0];

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Control speed */
speed_error = speed_ref - speed_estimated_ka;
Pprop = (speed_error)*Kp;

```

```
speed_error_int = speed_error_int + (speed_error)*DT*Ki;

current_PI = Pprop + speed_error_int; /* PI controller output*/

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* To limit the current reference to 15A peak current*/

if(current_PI>current_max)current_PI=current_max;
        else if(current_PI < -current_max) current_PI = -
            current_max;
        else current_PI = current_PI;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Current Error */

current_error = current_PI - current_loop;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* second saturation for safety */
if(current_error>current_max) current_error =current_max;
        else if(current_error< -current_max) current_error = -
            current_max;
        else current_error = current_error;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Shift current range 0-30*/
/* Piggy output */

Volt_tmp = (current_error*2.425)/17;

Volt_out = 2.425 - Volt_tmp;


/* third saturation for safety */
if(Volt_out >3.5) Volt_out =3.5;
        else if(Volt_out < 1.5) Volt_out  = 1.5;
        else Volt_out  = Volt_out ;

```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
/* Segnals output */

ds1102_da(1,speed_estimated_ka/1000);
ds1102_da(2,speed_estimated_mb/1000);
ds1102_da(3,speed_filt/1000);
ds1102_da(4,Volt_out/10);

                 exec_time = time_elapsed(0, count0);

     end_isr_t0();
}
/*-------------------------------------------------------------*/
/*  Main Function */

main()
{
 int i=0;

 init();                     /* initialize hardware system */
 *error = NO_ERROR;          /* pg 63 intialize error flag */

  start_isr_t0(DT);

/*  clearing the counter */
ds1102_inc_clear_counter(1);


err_cnt = 0;
CHECKERR:
 while (*error == NO_ERROR);                      /* background
    process */
 *error = NO_ERROR;

 init();                              /* initialize hardware
    system */
 start_isr_t0(DT);
 err_cnt = err_cnt + 1;
 goto CHECKERR;
}
```

Listing A.2: KAFeedback.c

Bibliography

[1] S. Jeon and M. Tomizuka, “Benefits of acceleration measurement in velocity estimation and motion control,” *Control Engineering Practice*, vol. 15, pp. 325–332, March 2007.

[2] S. J. Kwon, W. K. Chung, and Y. Youm, “A combined observer for robust state estimation and kalman filtering,” *Proceedings of the American control conference*, pp. 2459–2464, 2003.

[3] H. W. Kim and S. K. Sul, “A new motor speed estimator using kalman filter in low speed range,” *IEEE Transactions on Industrial Electronics*, vol. 43(4), pp. 498–504, 1996.

[4] S. H. Kim and S. K. Sul, “An instantaneous speed observer for low speed control of ac machine,” *Proceedings of the IEEE applied power electronics conference and exposition*, vol. 2, pp. 581–586, 1998.

[5] D. J. Lee and M. Tomizuka, “State/parameter/disturbance estimation with an accelerometer in precision motion control of a linear motor,” *Proceedings of the 2001 ASME IMECE*, vol. DSC-24578, 2001.

[6] M. T. White and M. Tomizuka, “Increased disturbance rejection in magnetic disk drives by acceleration feedforward control and parameter adaptation,” *Control Engineering Practice*, vol. 5(6), pp. 741–751, 1997.

[7] B. O’Callaghan and D. R. Kavanagh, *Development of Rotary Accelerometer using MEMS Accelerometer Project Report*, Department of Electrical and Electronic Engineering University College Cork, 31 March 2008.

[8] Analog-Device, *ADXL210E Datasheet. Accelerometer with Duty Cycle*, supplyed directly in http://www.analog.com.

[9] http://www.memagazine.org.

[10] DataSheet-Moog, *Brushless Servomotor G400 Series*, 2008.

[11] Moog-Manual, *DS2100 Digital Controller. Installation and User's Manual*, supplyed in the website http://www.moog.com/literature/ICD/ds2100servodrives-um.pdf.

[12] M. K. Hyunchul Shim and M. Tomizuka, "Use of accelerometer for precision motion control of linear motor driven positioning system," *Industrial Electronics Society. Proceedings of the 24th Annual Conference of the IEEE*, pp. 2409–2414, 1998.

[13] dSPACE Manual, *Floating-Point Controller Board, DS1102.*

[14] S. Komada, K. Ohnishi, and T. Hori, "Hybrid position/force control of robot manipulator based on acceleration controller," *Proceedings of the IEEE 1991 international conference on robotics and automation*, pp. 48–55, 1991.

[15] B. Wildrow, "Statistical analysis of amplitude-quantized sampled data systems," *AIEE Transactions on Application and Industry*, vol. 81, pp. 555–568, 1961.

[16] J. G. O'Donovan, *Control and Estimation Strategies for Nonlinear Motor Drive Systems Ph.D Thesis*, Department of Electrical Engineering and Microelectronics University College Cork, June 1997.

[17] E. D. Tung, Y. Urushisaki, and M. Tomizuka, "Low velocity friction compensation for machine tool feed drivers," *Proc. American Control Conf.*, pp. 1910–1914, 1993.

[18] U. Schafer and G. Brandenburg, "Compensation of coulomb friction in industrial elastic two mass systems through model reference adaptive control," *Proc. European Conf. Power Electronics and Applications (EPE)*, pp. 1409–1415, 1989.

[19] P. E. Dupont and E. P. Dunlap, "Friction modelling and control in boundary lubrication," *Proc. American Control Conf.*, pp. 1910–1914, 1993.

[20] S. Bolognani, *Azionamenti Elettrici Course Notes*, Department of Electrical and Electronic Engineering, University of Padova.

[21] W. Chen and M. Tomizuka, "Estimation of load side position in indirect drive robots by sensor fusion and kalman filtering," *AACC American Control Conference*, pp. 6852–6857, 2010.

[22] J. Corres and P. Gil, "Instantaneous speed and disturbance torque observer using nonlinearity cancellation of shaft encoder," *IEEE Transactions on Industrial Electronics*, vol. 4, pp. 540–592, 2002.

[23] Z. Zedong, L. Yongdong, M. Fadel, and X. Xi, "A rotor speed and load torque observer for pmsm based on extended kalman filter," *IEEE International Conference on Industrial Technology*, pp. 223–238, 2006.

[24] P. Meehan and K. Moloney, *Basic Principles of Operation and Applications of the Accelerometer Report*, Limerick Institute of Technology.

[25] dSPACE Manual, *ds1104 R&D Controller Board, Installation and Configuration Guide.*

[26] ——, *MLIB/MTRACE, MATLAB-dSPACE Interface and Trace Libraries.*

[27] ——, *Real-Time Interface (RTI and RTI-MP), Implementation Guide.*

[28] ——, *Control Desk, Experiment Guide.*

[29] ——, *Control Desk, Automation Guide.*

[30] ——, *Connector Panels and LED Panels. CP1102/CLP1102.*

[31] ——, *DS1102 Software Environment.*