UNIVERSITY OF OSLO
Department of Physics

An Attitude Detumbling System for the CubeSTAR Nano Satellite

Master thesis

Kjetil Rensel

August 15, 2011

Abstract

This thesis describes the first hardware implementation of the Attitude Determination and Control System (ADCS) of the CubeSTAR satellite. CubeSTAR is a student satellite under development at the University of Oslo. A module card has been designed, consisting a magnetometer, gyro sensors, magnetorquer control, and a microcontroller. The complete hardware developed, is the necessary hardware to perform detumbling of the satellite, and will be a part of the complete ADCS. Two different MEMS gyro sensors models have been compared to each other, and a calibration process has been developed. The magnetometer has also been calibrated, utilizing an ellipsoid fitting method. A production method for magnetorquers has been developed, by making a custom made coil winding machine.

Acknowledgments

This master thesis is the fulfilling work of the Master of Science in Electronics and Computer Technology at the Department of Physics, Faculty of Mathematics and Natural Sciences, University of Oslo, Norway. The thesis is written in the period during August 2010 to August 2011, under the supervision of Associate professor Torfinn Lindem.

The digital PDF version of this document includes a lot of nice features, which is not possible to achieve on paper. All cross references and content lists in the document can be clicked on to go to the referred object. The Bibliography contains back-references, and all references freely available on the Internet are accessible by clicking on the reference title. Since there are many active objects, the links is not marked, but is visible by placing the cursor on them. Many of the figures, included C Block Diagrams, Schematics PCB and Part List are vector graphics which can be zoomed in to get all details without loss of quality. All figures without a reference are created by the author, and can freely be utilized.

I would like to thank Torfinn Lindem for his support during the work, and for his work on the CubeSTAR project. I also want to thank all the guys at the mechanical workshop and the electronic workshop for supporting with all kind of questions. Especially I want to thank Daniel Bakke Randby, Thor Arne Agnalt and Steinar Skaug Nilsen at the mechanical workshop for the great cooperation in designing the mechanical components for my thesis. Together we made a good team. Thanks to Sensonor which was kind and supported me with three high precision gyro sensors. As a nice start of the master thesis period, I got the opportunity of participate on the annual Small Satellite Conference in Logan, USA. I want to thank Tore Andre Bekkeng and Jørn Inge Aasen for a great trip, and NAROM for supporting it economically. At last I want to thank my beautiful girlfriend Helen Jarnes and my fantastic family for all support.

Oslo, August 2011

Kjetil Rensel

Contents

List of Figures

List of Tables

Nomenclature

ADC	Analog to Digital Converter
ADCS	Attitude Determination and Control System
AMR	Anisotropic Magnetoresistance
ASCII	American Standard Code for Information Interchange
ASIC	Application-specific integrated circuit
CCD	Charge-coupled device
CMOS	Complementary metal–oxide–semiconductor
CoCom	Coordinating Committee for Multilateral Export Controls
ECEF	Earth Centered, Earth Fixed
ECI	Earth Centered Inertial
emf	Electromotive force
EPS	Electrical Power System
ESTEC	European Space Research and Technology Centre
FOV	Field of View
FSM	Finite State Machine
GCC	GNU Compiler Collection
GPS	Global Positioning System
GUI	Graphical User Interface
I^2C	Inter-Integrated Circuit
IC	Integrated Circuit
IGRF	International Geomagnetic Reference Field
LCC	Leadless Ceramic Carrier
LCD	Liquid Crystal Display

LEO	Low Earth Orbit
m-NLP	Multi-Needle Langmuir Probe
MEMS	Microelectromechanical Systems
NAROM	Norwegian Centre for Space-related Education
NTNU	Norwegian University of Science and Technology
OBDH	On-Board Data Handling
P-POD	Poly-PicoSatellite Orbital Deployer
PCB	Printed circuit board
PDI	Program and Debug Interface
PWM	PulseWith Modulation
SCL	Serial Clock
SDA	Serial Data Line
SPI	Serial Peripheral Interface Bus
SVD	Singular value decomposition
TWI	Two-Wire Interface
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VI	Virtual Instrument

Chapter 1

Introductions

The CubeSTAR satellite is a $2kg$ nano sized satellite, complying with the CubeSat standard. CubeSTAR is being fully developed at the Department of Physics, University of Oslo (UiO). A Multi-Needle Langimur Probe for detecting electron density in ionospheric plasma has been developed, and is the payload of the satellite. The main goals of the CubeSTAR project is additional to testing and receiving data from the Langimur Probe, to provide interesting master thesis for the students. Several thesis have already been completed, and several are yet to come.

When the satellite is being deployed from the launch rocket, it will possess an undesired angular velocity, referred to as spin. The spin has to be reduced, and control of the satellite's attitude has to be achieved. This thesis will be focusing on implementing sensors and actuators, and implement a spin reducing algorithm. The system will be a part of the total Attitude Control and Determination System (ADCS) which is going to be further developed in the ongoing CubeSTAR project.

1.1 CubeSat Standard

The CubeSat project was initiated in collaboration between California Polytechnic State University (Cal Poly) and Stanford University's Space System Development Laboratory in 1999. It was initiated to provide a standard platform for nano satellites, including mechanical and electrical specifications. A Poly-PicoSatellite Orbital Deployer (P-POD) for CubeSats was also developed. The P-POD is a device carrying the satellites onboard the launch rocket, and taking care of the deployment of the satellite. It is designed to make it easy and secure for launch providers to include several CubeSat satellites as addition to the main payload on a rocket. The standardization makes launch of a CubeSat satellite relatively cheap, compared to a non-standard launch. Today more than 100 universities, high schools and private firms are collaborating and sharing information in the CubeSat community. The complete set of CubeSat specifications are found in [1]. A one unit (1U) CubeSTAR satellite is a $10cm$ cube, with a maximum weight of $1.33kg$. Bigger CubeSat satellites can be achieved by adding up units in the height direction. The dimensions of a 2U CubeSat is $10cm * 10cm * 20cm$. The standard also states that the center of gravity have to be located within a sphere of $2cm$ from the geometric center. A common understanding of the naming scheme is that a satellite weighing $1kg$-$10kg$ is classified as a nano satellite, and a $0.1kg - 1kg$ pico satellite. A $1U$ CubeSat may then fit either of these classifications, while CubeSTAR is a nano satellite. As a result of the

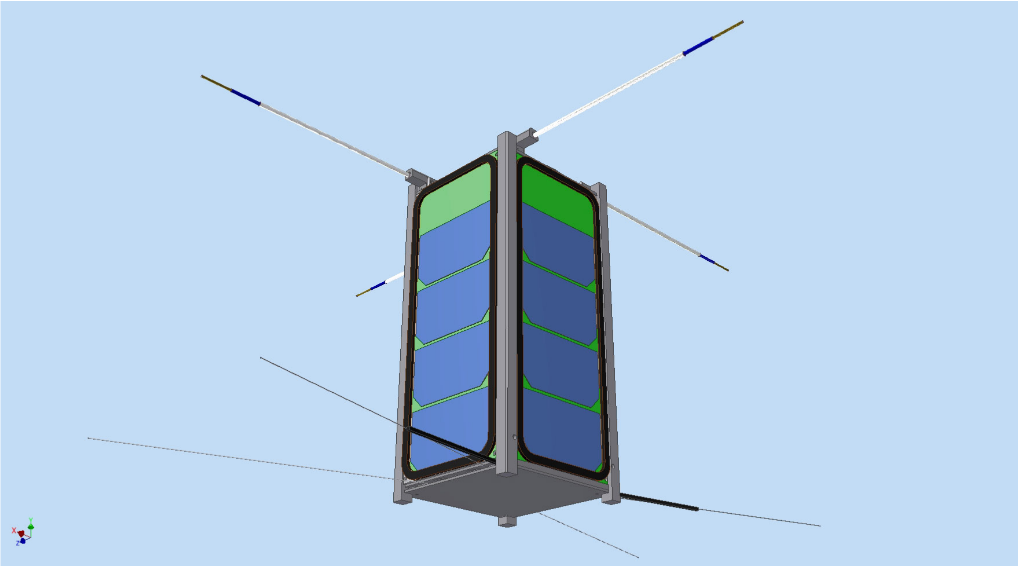


Figure 1.1: CAD drawing of the CubeSTAR satellite with Langimur Probes (upper) and antennas deployed. Each of the four long-sides contains solar cell panels. In this illustration, two magnetorquers are surrounding the solar cell panels. Credit: The Mechanical Workshop, Dpt. of Phy.

popularity of CubeSat, several companies have specialized on developing subsystems and satellites for the CubeSat standard. Most components for building a CubeSat can be purchased on web shops on the Internet.

1.2 CubeSTAR Project

The CubeSTAR project was initiated in collaboration between UiO and the Norwegian Center for Space-related Education (NAROM). Since one of the main goals of the project are the development of the satellite in itself, it is desired to develop as much as possible of the satellite locally at the University.

CubeSTAR is a 2U CubeSat with fixed solar panels on each of the four long sides, communication antennas in one end, and Langimur Probes in the other end. Inside the satellite, a backplane card is connecting together several module cards, solar panel cards and the battery pack. Figure 1.1 shows a preliminary CAD drawing of the satellite, the way we think it is going to look like. The CubeSTAR project is considered as being built up of the following subsystems:

- Payload [2]
- Electrical Power System (EPS)[3]
- Attitude Determination and Control System (ADCS)[4]
- On-Board Data Handling (OBDH)
- Communication [5], [6]

In addition to the subsystems of the satellite, a ground station [7] for communication is developed. Five thesis referred to in the text above is, as of August 2011, completed on the CubeSTAR project, while the payload thesis was originally a part of the ISI-2 rocket project. In addition to the student thesis, the structure has been developed by the mechanical workshop and a backplane by the electronic workshop, both at the Department of Physics.

1.2.1 Mission

The satellite will be deployed in an orbit altitude at $300 - 800km$, decided by the main payload of the launch rocket. Orbits in the altitude range up to $2000km$ are classified as a Low Earth Orbit (LEO). A satellites in LEO typically makes one revolution around the Earth in about $90min$. After the satellite has been deployed, it will be observable from the Earth with radar. Based on radar observations, we can calculate the position of the satellite at any given time. The parameters describing the orbit can also be transmitted to the satellite. Since a satellite in LEO is close to the earth, it will encounter an atmospheric drag. The drag is due to gases colliding with the satellite, slowly decreasing its velocity, which again leads to a change in the altitude of the satellite. Eventually the satellite will enter the atmosphere and burn up. Because of this, the altitude of the initial deployment will greatly impact the satellites length of life. The time from deployment until it burns up is expected to be several years. CubeSTAR is going to obtain a polar orbit, the satellite will be crossing close to each pole and has an inclination close to $90°$ to the equator. The polar orbit makes it possible to do measurements in the north areas of the earth on every revolution, which is the area of interest for our payload.

1.2.2 Scientific Payload

The payload of the CubeSTAR satellite is a Multi-Needle Langmuir Probe (m-NLP), design to measure electron density in the ionospheric plasma [2]. The m-NLP developed, measures the density at a resolution down to meter-scale, which is an improvement of. The information about the ionospheric plasma density over the polar cusps is of interests for space weather monitoring and to improve communication and navigation. The m-NLP is developed at the University of Oslo, and has been tested at ESTECs Plasma Lab and flown on the ICI-2 sounding rocket. It is of great interest for the m-NLP project to verify the system on a satellite. Experiences from flying on CubeSTAR will increase the interest for using the system to other projects.

1.3 Attitude Determination and Control

When the satellite is deployed from the rocket, an unwanted angular velocity is likely to be present. The forces acting on the uncontrolled satellite are not able to stop this rotation. For our main payload to success, the Langimur Probes have to be in front relative to the direction of velocity. If the probes are not in front, the measurement will be in electron turbulence from the satellite body moving through the ionosphere. It is thereby clear that we need to control the attitude of the satellite. The ADCS will be realized with sensors, actuators and computational power. The ADCS are planned to function in of two different modes, detumbling mode and Attitude Determination and Control mode (ADC-mode). The detumbling mode is a simple algorithm which only task is to reduce angular velocity. The detumbler mode will only be active when angular velocity is exceeding a given limit. Attitude determination and control is a more advanced control method, not active when in detumbler mode. A 10° attitude accuracy is set as a desired accuracy goal for the total control system.

1.4 Previous Work

CubeSat satellites have been launched around the world for nearly 10 years, which some of them have been developed in Norway. NCUBE1 (also named Rudolf) and NCUBE2 was designed at the Norwegian University of Science and Technology, NTNU, in collaboration with other educational institutions including UiO[8]. Unfortunately neither of the satellites became operational, because of problems during launch (NCUBE1) and probably deploying problems (NCUBE2). Even if some CubeSats fails, many have been several years in successful operation, like the Japanese CubeSat XI-V, which currently have been active in six years, and in a period were sending pictures of the Earth automatically to its own Twitter account. Since many Universities are basing their CubeSat development on student thesis, a lot of master theses are available on the subject. Aalborg University in Denmark did early develop a CubeSat, and their first launch, AAU-Cubesat 1 [9, 10] was performed in 2003.

In Wertz (1978) [11], many of the basic principles of Attitude Determination and Control in Space are thoroughly explained. The book is probably the most cited source when it comes to this particular subject. A more practical overview of physical subsystems of a spacecraft is presented in [12], while [13] describes control systems and spacecraft dynamics. An excellent beginner's guide in Spacecraft Dynamics and Control[14] is written as a compendium for a course at Virginia Tech, and have to be mentioned, despite the fact that the guide is not officially published.

1.4.1 Relevant Work on the CubeSTAR Project

At this time, one Master's thesis is being completed on the ADCS subsystem of the CubeSTAR. In the thesis [4], simulations of a purposed control system is performed. Simulation includes comparison between a Proportional-Derivative and a Linear-Quadratic Regulator, verification of the b-dot detumbler controller, and simulation of the uncontrolled satellite. A proposal of design parameters for magnetorquers is also presented. In this work, no work on the attitude determination is performed, and a given, error free attitude is assumed. The representation of the b-dot controller and the magnetorquer calculations is a basis for the further work on these subjects in this thesis.

Additional to the work being done on the ADCS, the electrical backplane, the module card standard and the mechanical structure, is relevant for this thesis. The electronic being produced in this thesis must comply with these components. The detumbling b-Dot control law

1.5 Goals of the Thesis

This thesis is a part of a ongoing project, and it is important for the project that the thesis is bringing the project forward. It is necessary for future work on the ADCS system to have a hardware basis with sensors and actuators. This thesis goal is to make all the necessary hardware available for detumbling, but also keep in mind the purpose the hardware is going to serve in the Attitude Determination and Control System. Generally, the following tasks should be performed:

- Design a module card for the satellite, wich is going to be the first version of the ADCS card.
- Implement the necessary hardware for a detumbling process.
- Provide good measurement data to the rest of the subsystems.

Summarized, the most important is to maintain a good progression of the CubeSTAR project, by developing the ADCS system.

1.6 Outline of the Thesis

Chapter 2 Attitude Determination and Control System A short introduction to attitude representation. Common sensors and actuators are presented, included those utilized in this thesis.

Chapter 3 Magnetorquers A fully overview of the theory, design and production of the Magnetorquer. The design of a coil winder is also a part of this chapter.

Chapter 4 Electronic Design Description of the electronics designed. This includes hardware, firmware and its accompanying PC-software.

Chapter 5 Sensor Calibrating Calibration methods is presented for the gyro sensors and the magnetometer.

Chapter 6 Discussion A conclusion of the work done, along with a proposal of future work.

Appendix A Coil Winder User Manual A step-by-step user manual of how to use the coil winder, and how to modify it to do other dimensions of a coil.

Appendix B Schematics PCB and Part List Printout of all schematics and PCBs for the ADCS card, the Mini Backplane card and the coil winder card.

Appendix C LabView Source Code Printouts of the Front Panel and the Block Diagram of the LabView VI designed.

Appendix D Microcontroller Source Code Printouts of all source code developed for the microcontrollers.

Appendix E Matlab Source Code Printouts of matlab source code utilized in calibration process

Appendix F CD A CD is attaced in the paper version of the thesis

Chapter 2

Attitude Determination and Control System

The orientation of a spacecraft in space is called its attitude. Most spacecrafts have some instruments or antennas which have to be pointed in a specific direction. To achieve this, control of the attitude is desired. Control of the spacecraft can be implemented by passive methods or an active Attitude Determination and Control System (ADCS). In this chapter the attitude is defined, and a method of representing it is described. Common methods, sensors and actuators earlier utilized on CubeSat projects are presented.

2.1 Attitude Representation

2.1.1 Reference Frames

A *reference frame* is a three dimensional Cartesian coordinate system, normally fixed to an object, like a spacecraft or a planet. The axes of a reference frame fixed to a rigid object, are normally defined to the logical directions of the object itself. A representation of the attitude between two reference frames (or objects) is achieved by the rotation matrix between them 2.1.2 . A reference frame is a three dimensional Cartesian coordinate system denoted by $\mathcal{F}_b$. Its triad of unit vectors is denoted $\hat{\mathbf{b}} = \left\{ \hat{b_1} \quad \hat{b_2} \quad \hat{b_3} \right\}^T$, where b is a suitable letter of the reference frame represented. The $\hat{}$ denotes unit vectors. In order to describe and analyze attitude dynamics, several reference frames have to be defined.

Satellite Body Frame

The satellite body frame, $\mathcal{F}_b$, has it origin in the mass center of the satellite. The respective body frame axis is aligned in the same directions as the satellites mechanical axis. Where the x_b axis is pointing in the forward direction, the z_b axis is pointing in what is defined as down direction on the satellite, and y_b axis completes the Cartesian right-hand rule. The rotation of a spacecraft about the axis x_b, y_b and $z_b$are respectively named roll, pitch and yaw.

Satellite Orbit Frame

The satellite orbit frame, $\mathcal{F}_o$, shares its origin with the body frame. The x_o axis points in the velocity direction of the orbit, while the z_o axis points nadir (towards the Earths center). The y_o axis completes the Cartesian right-hand rule. The satellite orbit frame is considered as the attitude reference for the body frame. *The attitude of the satellite is defined as the orientation of the satellite body frame in reference to the orbit frame.*

Earth Centered Inertial frame

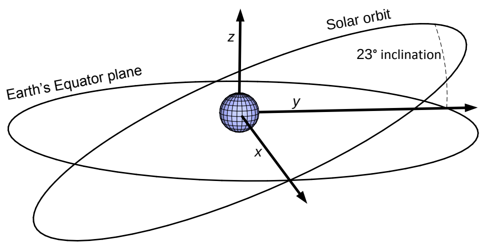


Figure 2.1: In the ECI frame, the earth is fixed at origin, and the sun is orbiting the Earth. The ECI frame axis are illustrated.

The Earth Centered Inertial (ECI) frame , denoted $\mathcal{F}_i$, has its origin in center of the Earth. The z_i axis points toward the geographic north pole. The x_i axis points toward the vernal point in the ecliptic coordinate system., which is the point where the sun is at on March equinox. The y_i axis completes the right hand rule. For Newton's laws to be valid, a non accelerating (inertial) frame is required. The ECI frame possesses this property.

Earth Centered, Earth Fixed frame

The Earth Centered, Earth Fixed (ECEF) frame, denoted $\mathcal{F}_e$, shares z_e-axis with ECI, pointing toward the geographic North Pole. The x_e-axis points toward the 0°latitude and 0° longitude point. As the name states, ECEF is fixed to the Earth, and its surface.

2.1.2 Rotation Matrix (Directing cosine matrix)

A rotation matrix $\mathbf{R}$, is a 3×3 transformation matrix, able to rotate a vector or express the rotation between two vectors. If the two vectors are reference unit vectors, $\mathbf{R}$ is the rotation between the reference frames. A vector represented in a reference frame $\mathcal{F}_a$, denoted $\vec{v}_a$,is represented in another reference frame $\mathcal{F}_b$ by

$$\vec{v}_b = \mathbf{R}^{ba} \vec{\mathbf{v}}_a$$

The superscript of $\mathbf{R}^{ba}$denotes the rotation from $\mathcal{F}_a$ to $\mathcal{F}_b$. Each superscript letter is close to the corresponding vector in an equation. Recall the satellite attitude definition as $\mathcal{F}_b$ represented in $\mathcal{F}_i$, hence the attitude is the rotation matrix $\mathbf{R}^{ib}$. The rotation matrix is a rotation matrix if, and only if

$$\mathbf{R} = \in SO(3)$$

where $SO(3)$is defined as

$$SO(3) = \begin{matrix} \mathbf{R} \mid \mathbf{R} \in \mathbb{R}^{\mathbf{3}\times\mathbf{3}}, \\ \mathbf{R}^T\mathbf{R} = \mathbf{I}, \\ det\mathbf{R} = 1 \end{matrix}$$

Because $\mathbf{R}$ is in $SO(3)$, we have that

$$\mathbf{R}^{ba} = \mathbf{R}^{ab^T} = \mathbf{R}^{ab^{-1}}$$

2.2 Sensors

A variety of sensors have been used on spacecrafts through the years. Some common sensors are presented here. Since most of the sensor output have to be compared to corresponding mathematical model, some considerations of the model is also included.

2.2.1 Magnetometer

A magnetometer is a sensor measuring the magnetic field vector. Magnetic sensors in different variations are the most common navigation sensors utilized. The Earths magnetic field is well defined and relatively strong in the altitude of Low Earth Orbit. This makes it well suited for attitude purposes, and is present on all CubeSat projects known to the author. A three-axis measurement is required, and was traditionally implemented by combining several one- or two-axis analog sensors, with additional necessary circuitry. The development of consumer navigation units, including smartphones, have led to small, power effective and cheap three-axis magnetometers. These magnetometers include most of the circuitry on the chip, including the ADC.

The Earth's magnetic field, which the magnetic measurement must be compared to for an attitude to be determined, is close to a magnetic dipole. A mathematical dipole model is normally not accurate enough, and a more accurate model named the International Geomagnetic Reference Field (IGRF) is common to implement. IGRF is a standardized mathematical model of the Earth's magnetic field, with a precision of one tenth of an nT. The model is a 13th order formula, with a disadvantage in its high requirements in computational power, compared to the dipole model.

2.2.1.1 Honeywell HMC5883L

In this thesis, a Honeywell HMC5883L magnetometer is being used. The magnetometer is based upon an Anisotropic Magnetoresistance (AMR). AMR occurs in ferrous materials, which changes its resistance when a magnetic field is applied perpendicular to the current flow. The resistive strips are connected together as a Whetstone Bridge with the strips as the four variable resistors. The resistors are all placed in the same direction, but connected so that the current is flowing in different directions. In that way, the same applied magnetic field will cause the resistance to increase in two of the resistors, and decrease in the other two. When applying a voltage, V_b, over the Whetstone bridge, an output voltage linear to the applied magnetic field is present. The changes in output voltage is given by

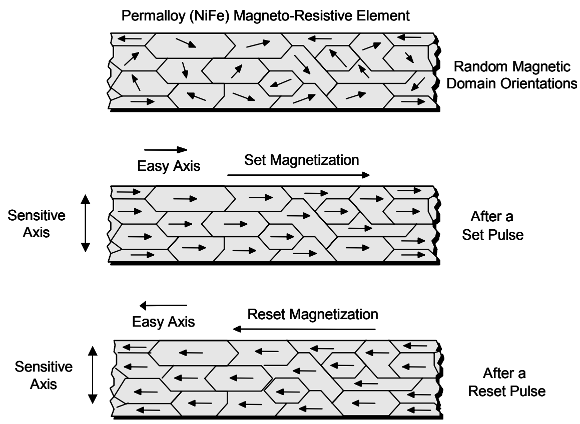


Figure 2.2: Magnetic domains inside a Permalloy (NiFe) Magneto-Resistive Element [18]

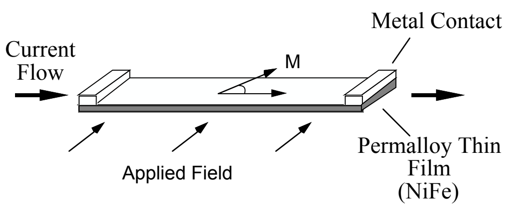
$$\Delta V_{out} = \left(\frac{\Delta R}{R}\right) V_b$$

$$\Delta V = SHV_b$$

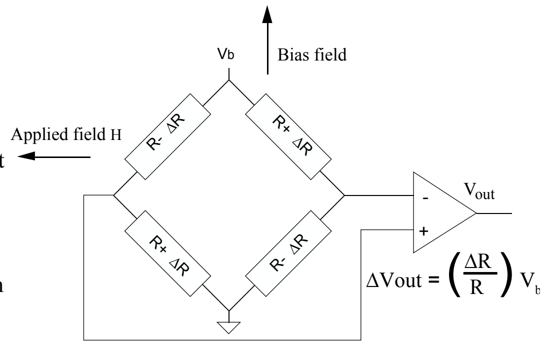
where

$$S = 3\frac{mV}{V/Oe}$$

The magnetoresistive sensors in HMC5883L are fabricated with Permalloy (NiFe) thin films. The sensor elements consist of small magnetic domains, each with a magnetic orientation. Similar to a magnetic tape for storage, the orientations could be permanently changed by a magnetic field of sufficient strength. A smaller magnetic field will only temporarily change the magnetic domains. When the magnetic domains are aligned in the same direction, the change in resistance is following the angle between the direction of magnetization and the current flow. In operation, the current is constantly flowing in the same direction, but the external magnetic field is changing in strength and direction affecting the resistance. The external measured field is normally only temporarily changing the magnetic domains. It is important that the permanent magnetization is all the same direction, perpendicular to the current flow. To achieve the correct permanent magnetization, it is required to be able to set this before operation. Magnetization is achieved by a *set/reset* circuit. A set/reset circuit is able to apply a strong magnetic field of above $4mT$. A *reset* is when an inverted magnetic field is applied. Reset orientates the magnetization in the opposite direction of a *set,* which leads to an inverted output response from the sensor. In HMC5883L the complete set/reset circuit is embedded in the chip, and automatically performed.



(a) The magnetoresistive element's resistance varies with the magnetic field applied to it.



(b) Magnetometer Wheatstone bridge

Figure 2.3: Magnetoresistive sensor properties [17]

2.2.2 Gyroscopic Sensor

Gyroscope, commonly shortened gyro, is a device maintaining or measuring orientation. Gyroscopes are based on the principles of conservation of angular momentum. A classical mechanical gyro, is a rotating mass, mounted free to move in all directions. If the base of the gyro is being tilted, the rotating axis will tend to maintain its orientation. Because of this, gyroscopes are said to be a "keeper of direction". Another property of a gyro, is the ability to get an output torque proportional to an angular velocity. The rotating mass must be mounted with only one axis free to rotate, perpendicular to the spinning axis. This axis is considered as the output axis, where a torque proportional to the angular input velocity could be observed. The input axis is perpendicular to both the spinning axis and the output axis. This effect is called the precession of a gyro, and can be explained by Newton's law of motion for rotation: *The time rate of change of angular momentum about any given axis is equal to the torque applied about the given axis.*[19]

The classical gyro are big mechanical constructions, and inappropriate for a CubeSat. Fortunately the principle of a gyro is implemented in various mechanical constructions, including microelectromechanical systems (MEMS). MEMS is the technology of electricity driven mechanical constructions in scale of μm. A MEMS gyro has a vibrating mass instead of a rotating one. The last couple of years, MEMS gyros have been cheaper and smaller, and like magnetometers, the development have been driven by consumer electronics. A three-axis magnetometer with digital output is available in small IC packages.

Since a gyro only gives us the angular velocity, integration must be done in order to obtain the attitude. A drawback with the integration is that a bias on the gyro will add up. Because of this, the gyro is not suited for obtaining an absolute attitude, but just to add additional accuracy to the rest of the sensors in an ADCS. The complexity, price and size just a few years back, made gyros a rare sight in CubeSats. This is about to change.

2.2.2.1 Sensonor ButterflyGyro

In this thesis Sensonor SAR sensors are being used. The sensor chip contains a ButterflyGyro MEMS single axis gyroscope and a mixed mode Application-specific integrated circuit (ASIC) circuit. The MEMS construction principle consists of two butterfly shaped masses mounted together to each other and to the main structure via a structural beam. The surrounding structure, the beams and the butterfly masses are all fabricated as one single-crystal silicon part, with the flexibility for the butterfly masses to rotate slightly in all directions. Excitation and detection electrodes are placed underneath the butterfly structure. The excitation probes are forcing the masses to oscillate, but due to an asymmetric design of the bearing beams, most of the mass rotation is in the plane of the structure, and not in the direction of the force produced by the excitation probes. The horizontal mass oscillation applied by the excitation probes are called excitation motion. The excitation motion of the two butterfly masses are a small back and forth rotation, always in opposite direction to each other. When an angular rate around the input axis, horizontal and perpendicular to the beam, is present, the Coriolis forces are oscillating the masses around the detection axis in phase to the excitation motion. The outer butterfly "wings" are then oscillating up and down, changing the capacitance in the sensing probes in phase with the excitation motion, but with amplitude following the angular rate. The ASIC is measuring the angular rate via the capacitance. Because of the symmetry and double-side excitation and detection, the sensor has a low sensitivity to shock and vibrations. More detailed information about the principles of a Sensonor ButterflyGyro can be found in [25].

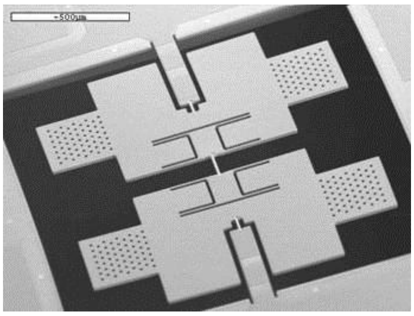


Figure 2.4: MEMS structure of a ButterflyGyro. The two butterfly spade masses are mounted in a thin asymmetric bearing, which makes the masses to also oscillate in the plane of the structure, even if the excitation forces are directly under the masses.

2.2.2.2 InvenSense ITG-3200

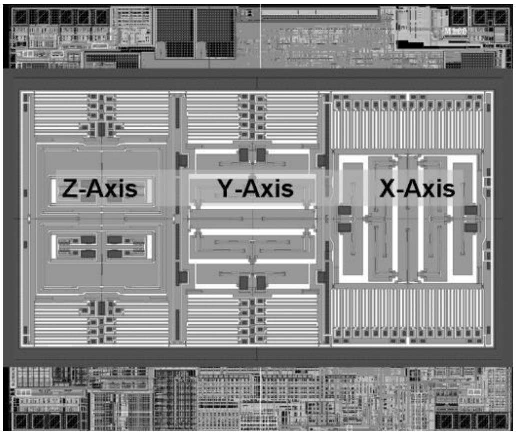
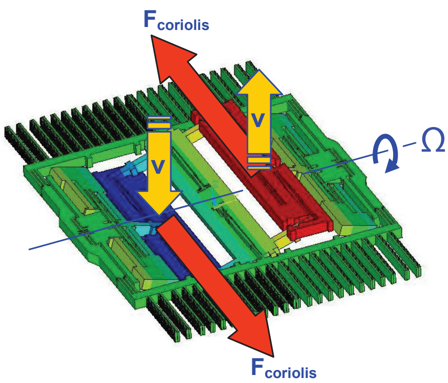
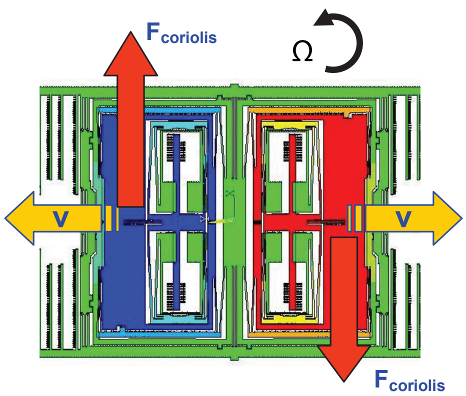


Figure 2.5: A photo of a three axis InvenSense gyro. The MEMS part is big compared to the surrounding ASIC circuitry. [15]

Additional to the Sensonor gyro sensors, the InvenSense ITG-3200 is utilized in this thesis. The sensor is a small three-axis gyro sensor developed for the consumer market. The ITG-3200 has a different structure of the moving MEMS masses compared to the SAR sensor. To be able to produce a z-axis gyroscope on the same die as the x- and y-axis gyros, the z-axis gyro also have to have a bit different working principle than the other two. For all of the three axis, there are a two mass system mounted so the forced mass movement is opposite to each other. When an angular velocity is present on the corresponding input axis, the Coriolis effect forces the mounting frame to twist in the plane of the frame and die. The mounting frame has a capacitive sensing structure outside of the frame, measuring the twisting of the frame. For the x- and y-axis, the masses are moving up and down, relatively to the die plane. In the z-direction, the masses are moving in the die plane apart and against each other. More detail information can be found in .

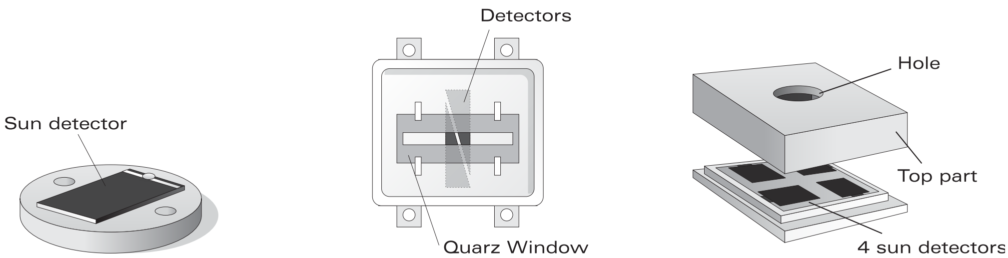


(a) 3D view of x- and y-axis



(b) Z-axis seen from directly above

Figure 2.6: Drawing of the MEMS structure inside the InvenSense ITG-3200 gyro sensor. To create all sensor axies on the same die, the Z-axis sensor has a slightly different principle than the x- and y-axis. [15]



(a) Single axis (b) Two axis (c) Three axis

Figure 2.7: Three axis, two axis and single axis sun sensor [9]

2.2.3 Sun Sensors

Sun sensors are measuring the direction to the sun, relative to the spacecraft body. The sensor is not operative in eclipse, which can be a significant time of every orbit. The sun sensor is commonly used because of its accuracy compared to its simplicity. The sensor can be analog or digital, measuring one, two or three axis. One axis analog sensors are normally based on photocells whose output current is proportional to cosine of the angle between the direction to the sun and the normal of the photocell. The output is given by

$$I(\alpha) = I(0)cos(\alpha) \tag{2.1}$$

From which α can be determined. To fully get the three dimensional vector pointing to the sun $\mathbf{s}_b$, at least three sensors needs to see the sun. A common setup is placing a one-axis sensor 2.7a on each of the six sides on the satellite. It is possible to achieve the same effect only utilizing the solar cells, which is often placed on each side of the satellite anyway. Unfortunately, using the solar cells as sensor is not very accurate. The current produced by the solar cells is varying with temperature, and may also change with time. The one axis sensor setup suffers from inaccuracy in 2.1 when $\alpha \to 90°$ and the Earth albedo error. Earth albedo is the Sun light reflected from the Earth, and is near 30% of the solar flux, which is a big disturbance source[20]. It is possible to model the Earth albedo, but this is complicated and hard to get precisely. A two or three axis sensor can be achieved by having two or more sensors covered by a slot 2.7bor hole 2.7c. The two and three axis sensors are more unaffected to temperature variations, and are less exposed to the Earth albedo. Digital three axis sun sensors have been constructed by replacing the sun detectors in the three axis version by a CMOS image sensor. A digital CMOS sun sensor achieves high accuracy, but is advanced to implement.

A sun sensor is measuring the sun vector $\mathbf{s}_b$, which have to be compared to the mathematical sun-model describing $\mathbf{s}_i$. The calculation of $\mathbf{s}_i$ is described by[14].

2.2.4 Star Sensor

Stars are the most accurate optical source for attitude determination. Stars are small and defined in size, and maintain a fixed position. A star sensor can determine the attitude with very high accuracy, but is fairly complicated to implement. A star sensor system have to be able to distinguish stars apart from each other, which can be performed by considering their magnitude, light spectra and position relatively to each other. Before digital image sensors were available, *image dissector tube* sensors were utilized, but required much bigger spacecrafts than a CubeSat. In image dissector tube sensors, the field of view (FOV) were narrow, and down to one star alone was identified by the magnitude and light spectra. Modern star sensors based on CCD or CMOS image sensors and effective on board computers are making it possible for CubeSats to carry a star sensor. These star sensors have a wider FOV and base their star identifying upon the positions of the stars relatively to each other. All star sensors must have an onboard computer carrying characteristics of a big selection of stars.

2.2.5 Earth Sensor

Normally an Earth sensor is designed to detect the horizon, often named horizon sensor. An earth sensor has high accuracy, but does not determine the attitude alone, and must be used in combination. The sensor can be an infrared sensitive photo detector, since The Earth emits like a uniform blackbody at a temperature about 290 K. Such a sensor works during eclipse, but has a low signal to noise ratio. Some earth sensors are utilizing the Earths albedo, which produces a large optical output. The classical horizon sensor basically generates a signal when sight of line of the sensor passes the horizon. Like for the star sensor, an earth sensor can be realized by an image sensor system. If a camera is not present on the satellite anyway, it is a normally not worth the effort to implement an earth sensor.

2.2.6 GPS

A GPS sensor can achieve the position of the satellite. An orbital position is necessary for the attitude determination, but is also achieved by observing the satellite from the ground. It is also possible to determine an attitude by measuring the phase variations in the GPS carrying signal at two antennas. Utilizing the GPS system in LEO is theoretically no problem, and can be fit into a CubeSTAR. A big challenge may be the restriction set by the Coordinating Committee for Multilateral Export Controls (CoCom). The GPS CoCom limit disables GPS devices to work when moving faster than $1852^{km}/_{h}$ or an altitude higher than $18km$, the limit is set to limit use of GPS in Intercontinental ballistic missiles. The limit is possible to overcome, and GPS receivers have been implemented in CubeSats. We do not consider a GPS receiver as useful enough for the CubeSTAR, compared to the complexity, power consumption and size.

2.3 Actuators and Passive Stabilization Methods

2.3.1 Gravity Gradient Stabilization

The gravity gradient stabilization method utilizes the Earths gravity. The gravity field is following the inverse-square law, which states that the strength of the gravity field is inversely proportional to the square of the distance from center of the Earth. A spacecraft with an uneven mass distribution, will tend to align its long axis to the field of gravity. For this method to be effective, a gravity gradient boom is applied. A gravity gradient boom is a relatively long boom which is will tend to align toward the earth. The boom is normally deployed from the main body of the satellite after deployment from the rocket. The gravity gradient stabilization is passive, which implies no power consumption, no software which can fail, and no dependency to sensors or determination.

In the other hand, a boom mechanism can be space and weight consuming. Additionally the stabilization is fixed and only in two dimensions.

2.3.2 Permanent Magnet and Hysteresis Rod

A permanent magnet mounted on the satellite will try to align to the Earths magnet field. This is an easy and reliable stabilization method, which is commonly utilized. Unfortunately, when passing the poles, the direction of the magnetic field is changing fast, and introduces tumble. Like on the gravity gradient method, the permanent magnet stabilization only works in two dimensions. The permanent magnet method is often combined with hysteresis rods. A hysteresis rod is made of soft magnetic material, which damps the rotation.

2.3.3 Magnetorquers

The most common attitude actuator on CubeSats, are magnetorquers. A magnetorquer is an electromagnetic coil, creating a dipole magnetic moment. The magnetic dipole, which is created perpendicular to the face area of a coil, will try to align to the Earths magnetic field. Normally three magnetorquers are placed perpendicular to each other, with the ability to set up a magnetic field in both directions. Such a configuration is able to fully control the attitude in all three dimensions. However, the torque created by each of the magnetorquer depends on the angle between the magnetic field and the magnetorquer. When the magnetic field is perpendicular to a coil, it does not create a torque at all. In such situations, the spacecraft is only controllable in two dimensions. A magnetorquer are being realized with or without a metal core, which again have influence on the size. It is also possible to realize a magnetorquer as traces on a multilayerPrinted Circuit Board (PCB). Magnetorquers are non-moving, easy to control, pretty small and the driving force is not being used up like on thrusters. The disadvantages are low accuracy and the control limitation when the magnetic field is perpendicular to a coil.

2.3.4 Momentum Wheels

A momentum wheel is a mass which can store angular momentum by rotating. The spacecraft body, including momentum wheels when implemented, is conserving angular momentum. To control the attitude of the spacecraft, the angular momentum is transferred to momentum wheels, by increasing rotation velocity. Momentum wheel are able to control the attitude with a very high accuracy. Since the angular momentum is still conserved in the spacecraft body, momentum wheels are implemented in combination with a momentum dumping actuator, like magnetorquers. Momentum dumping is also important, since the rotation is driven by an electric motor, which consumes power. Momentum wheels are implemented when high precision attitude control is necessary, but does require a lot of space and weight in a CubeSat.

2.3.5 Thrusters

Thrusters are utilizing Newton's third law, "The mutual forces of action and reaction between two bodies are equal, opposite and collinear". Thrusters are expelling propellant in the opposite direction of the generated force. To control attitude in three axes, at least six pairs of thrusters are required. The thrusters are placed in pairs canceling each others effect on the direction of movement. Thrusters have been tested on CubSats, but are not

Project name	**University**	**Year**	**Permanent magnet and hysteresis rod**	**Momentum wheel**	**Reaction wheel**	**Thrusters**	**Gravity gradient boom**	**Magnetorquer**	**GPS**	**Star tracker**	**Horizon sensor**	**Gyro**	**Accelerometer**	**Sun sensor**	**Magnetometer**
Cute-I	Tokyo Institute of Technology	2003										x	x	x	
CanX-1	University of Toronto	2003						x	x	x	x				x
DTUsat	Technical University of Denmark	2003						x			x			x	x
AAU Cubesat	Alborg University	2003						x						x	x
NCube2	NTNU	2005						x							x
XI-V	University of Tokyo	2005													
CUTE 1.7 + APD	Tokyo Institute of Technology	2006						x			x	x		x	x
ION	University of Illinois	2006				x		x						x	x
KUTEsat-1 Pathfinder	University of Kansas	2006						x						x	x
KuteSat 2	University of Kansas											x		x	
ICE Cube	Cornell University (New York state)	2006					x	x	x						x
SEEDS	Nihon University	2006										x			x
HAUSAT	Hankuk Aviation University	2006							x					x	
Ncube 1	NTNU	2006						x							x
CP2	California Polutechnic Institute	2006						x							x
CP1	California Polutechnic Institute	2006						x						x	
ICE Cube 2	Cornell University (New York state)	2006					x	x	x						x
Mea Huaka (Voyager)	University of Hawaii		x												
GeneSat-1	Center for Robotic Exploration and Space Technologies		x												
CP4	California Polutechnic Institute	2007						x							x
CSTB-1	The Boeing Company	2007						x						x	x
MAST	Tethers Unlimited	2007							x						
CP3	California Polutechnic Institute	2007						x							x
CAPE-1	University of Louisiana	2007	x												
Libertad-1	University of Sergio Arboleda	2007							x						
CanX-2	University of Toronto	2008		x		x		x						x	x
CUTE 1.7 + APD II	Tokyo Institute of Technology	2008													
Delfi-C3	Delfi University of Technology	2008	x											x	
AAUsat-2	Alborg University	2008		x				x							
Compass One	Fachhochschule Aachen	2008						x						x	x
Seeds 2	Nihon University	2008										x			x
Polysat CP6	California Polutechnic Institute	2009						x							x
AeroCube-3	Aerospace Corporation	2009	x								x			x	
Hermes	Colorado Space Grant Consortium	2009	x												x
BeeSat-1	Technical University of Berlin	2009			x			x				x		x	x
UWE-2	University of Würzburg	2009										x	x	x	x
ITUpSAT1	Istanbul Technical University	2009	x									x	x		x
AtmoCube	University of Trieste	2010							x						x
Goliat	University of Bucharest	2010		x					x						x
OUFTI-1	University of Liège	2010	x												
PW-Sat	Warsaw University of Technology	2010						x	x			x			x
SwissCube	Polytechnical School of Lausanne	2010						x				x		x	x

Table 2.1: Listing of a selection of previous CubeSat projects, an its Attitude Determination and Control components. The table is based on a web page [16] with partly inadequate source listings, but the information presented is as good as possible verified by further search on the Internet. The list should be considered on basis of its origin, but is still a good indication on how popular the different solutions are.

very suited for attitude control on such small spacecrafts. In addition to the thrusters itself, the propellant is taking up space and weight on the spacecraft. The propellant is a limited source, and the thrusters are only usable as long as propellant is left. Thrusters are better suited for bigger spacecrafts, and to change the direction of movement.

2.4 CubeSTAR ADCS

2.4.1 Sensors and Actuators Chosen

We decided to design an ADCS based on only magnetorquers as actuators. A magnetorquer based system is an obvious choice when we need control, but not with the accuracy a combination with momentum wheel would have given. The magnetorquers are simple to control, and does not have any moving parts.

As sensors for the system, we will in this thesis implement magnetometer and gyro sensors. Those are sufficient for the detumbling process, but do probably not give the desired accuracy for an attitude determination. An additional reference sensor, like a sun sensor, will probably be implemented in the future work of the project.

2.4.2 Attitude Determination and Control Mode

A complete magnetorquer based ADCS system, normally consists of two possible active modes, *Attitude Determination and Control* mode and *Detumbling* mode. The Attitude and Determination mode is active when the satellite is in operation, and does exactly what the name says. An attitude determination algorithm finds the optimal estimate of the attitude, based on all of the data available. The determination is often based on a Kalman filter, some times in combination with simpler sensor fusion algorithm of reference sensors like magnetometer and sun sensors. Determine the attitude is necessary to perform the control the satellite. The attitude controller algorithm controls the signal to the magnetorquers based on the estimated attitude, magnetic field and desired direction. An extra challenge for the controller is, as we are going to see in chapter 3, the satellite can never be turned around the axis parallel with the magnetic field. A great work on the CubeSTAR project have been performed on the control part by Stray.[4]

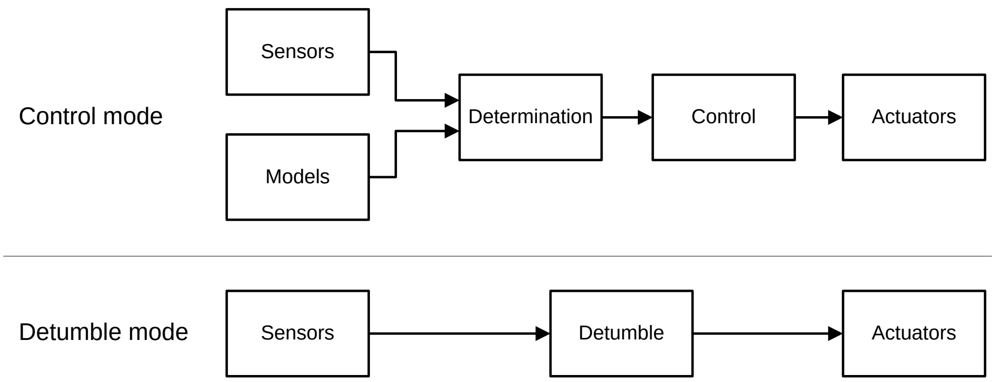


Figure 2.8: In Control mode the attitude is estimated and controlled. In detumble-mode, the attitude is unknown, but the angular velocity is lowered.

2.4.3 Detumbling Mode

The spacecraft is said to be tumbling when the angular velocity is exceeding a given value. The satellite is expected to tumble right after deployment from the P-POD, but can also enter a tumbling phase due to external disturbances and deployment of antennas. When the satellite is tumbling, a simple detumble controller is desired to reduce the angular velocity. A detumble controller is supposed to be simple and stable. A detumbler controller is preferably able to be implemented in a microcontroller. The simplicity of the controller is crucial, as it also works as a fallback system for the rest of the ADC mode. In some satellites, the attitude system must be fully operational for the payload to be useful, but in this project, a detumbled satellite not able to control is better than an uncontrolled one, since measurements probably still can be taken when the Langmuir probes is not in turbulence areas.

In [4] the common detumbler B-Dot was described and simulated for the CubeSTAR project. The B-Dot algorithm sets up a magnetic field on the magnetorquers, proportional to the derivative of the measured magnetic field. Since the B-Dot controller use the geomagnetic field as reference, the theoretical lowest angular velocity equals the change in the measured magnetic field.

The B-Dot controller is

$$\mathbf{m} = -K\dot{\mathbf{b}}_b$$

where $\mathbf{m}$ is the magnetic output moment, K is a positive constant gain and $\dot{\mathbf{b}}_b$ is the derivative of the measured magnetic field in the body frame

$$\dot{\mathbf{b}}_b = \mathbf{b}_b \times \omega_b^{ib} + \dot{\mathbf{b}}_i$$

which can be simplified to

$$\dot{\mathbf{b}}_b \approx \mathbf{b}_b \times \omega_b^{ib}$$

Simulations in [4] showed the ability to reduce angular velocity from $0.1\,\mathrm{rad/s}$ to $0.002\,\mathrm{rad/s}$ within 3 orbits with detumbling gain K set to 10000.

Chapter 3

Magnetorquers

In this chapter we are going to see how we can utilize magnetorquers to generate the desired control forces. The design process of the magnetorquers, and the coil winder machine, custom designed for production of the magnetorquers.

3.1 Magnetic Force in a Current Carrying loop

According to [21], a force **F** works on an electric conducting wire in a magnetic field **B** like

$$\mathbf{F} = i\mathbf{s} \times \mathbf{B} \tag{3.1}$$

where i is the current, and **s** is the length and direction of the wire. 3.1 can also be written in a non vector form representing the magnitude of **F**,

$$F = iBs \sin\theta \tag{3.2}$$

where θ is the angle between s and B.

We are now going to see how the magnetic forces work on a current-carrying loop. Figure 3.1 shows a $l * h$ sized rectangular loop with a current i flowing through it. We can not see the power source, but assume that there is one, and for now that it is just one single turn. The i arrow defines the current direction, opposite to the actual electron movement. The loop is lying in the x-z plane centered over origin, with a uniform magnetic field **B** parallel to the x-axis. Side 1 and 3 are also parallel to the magnetic field, and therefore does not have any magnetic forces working on them. Side 2 and 4 are perpendicular to the magnetic field, and the forces $\mathbf{F}_2$and $\mathbf{F}_4$ will be present. Since the amount of current and magnetic field is the same on both sides, the magnetic forces are equal in magnitude. According to 3.2, the magnitude of the forces are

$$F_2 = F_4 = ihB$$

The direction is according to the right hand rule, as in the figure.

The magnetic forces create a torque trying to rotate the coil around the z-axis. The magnitude of a torque generated by a pair of forces working perpendicular to a moment arm is given by [21]

$$T = |rF|$$

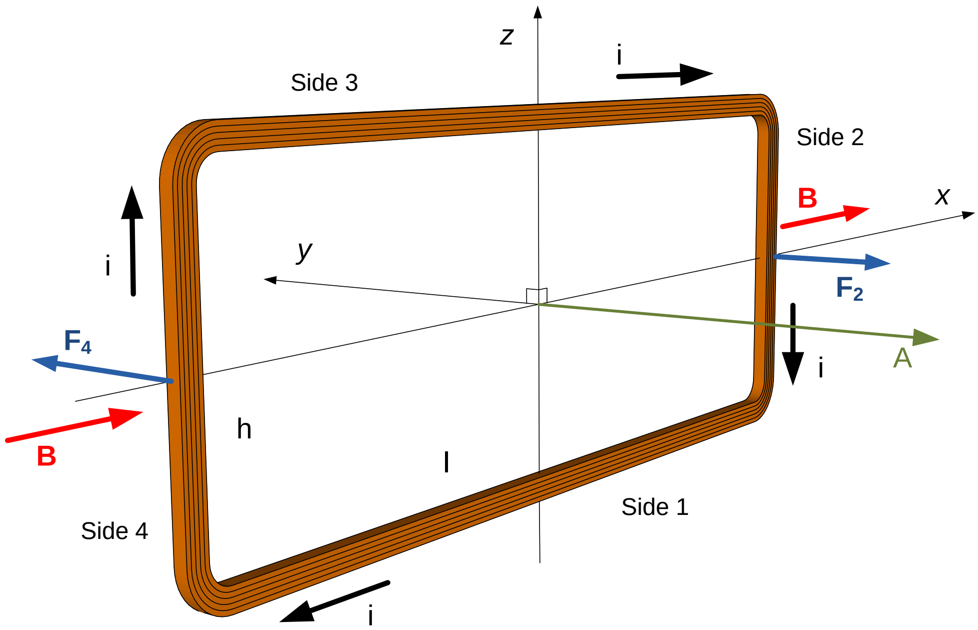


Figure 3.1: The magnetic forces, F, working on a current carrying loop in a uniform magnetic field.

where T is the torque and r is the distance between the forces. The torque created in our current can then be written

$$T = lihB$$

$$T = AiB$$

where A is the area. We have now removed the specific sides from the equation, and instead considering the area of our loop. If we make several turns of the wire in our loop, we can easily multiply the forces or the torque with the number of turns. By adding the turn number n, and represent the direction of the coil face with the area vector $\mathbf{A}$, we have for a air coil that

$$\mathbf{T} = ni\mathbf{A} \times \mathbf{B}$$

The magnetic fields tends to direct the face of a current-carrying loop toward the plane normal to the magnetic field.

The coil properties n, I, and A are defined as the magnetic dipole momentum μ,

$$\mu \equiv niA \tag{3.3}$$

The magnetic dipole momentum is a measure of the strength of an equivalent magnet, and is an important property when designing the coils for the CubeSTAR.Theory

The magnetic moment of a coil with air core is given by

$$\mu = nIA \tag{3.4}$$

where μ is the magnetic moment, I is the electric current through the coil and A is the face area of the coil.

We know that ohm's law states that $I = U/R$, where U is the voltage and R is the resistance of the coil. Since we do not want a voltage regulator for the magnetorquer, the voltage is not one of the variables we can control. In opposite, the resistance is affected as

$$R = \frac{nl\sigma(T)}{a_w} \tag{3.5}$$

where l is the circumference, a_w is the area of the cross sectional wire and σ is the material resistivity of the conductor, which is dependent of the temperature T. The material resistivity is approximately linear in our temperature range, and can be found by

$$\sigma(T) = \sigma_0[1 + \alpha(T - T_0)] \tag{3.6}$$

where $\sigma_0$is the resistivity at temperature T_0 and $\alpha$is the temperature coefficient of resistivity.

Combining 3.4and3.5 gives us

$$\mu = n\left(\frac{U}{\frac{nl\sigma(T)}{a_w}}\right)A$$

$$\mu = \frac{Ua_w}{l\sigma(T)}A \tag{3.7}$$

This shows that the magnetic moment is not affected of the total number of turns.

When designing the coil, we will have to calculate the weight, given by

$$m = nla_w\rho$$

where $m$is the total mass of the coil and $\rho$is the material density.

The power P consumed in the coil is given by

$$P = UI = I^2R$$

3.2 Design

3.2.1 Specifications

From [4] we have that a magnetic moment of between $60mAm^2$and $100mAm^2$is desired. A higher magnetic moment is not useful, since applying this can lead to instability. It is highly desired to use as little power as possible, since this is a valuable resource. An upper limit of the power consumption for each coil is set to $100mA$, but generally as low as possible. The material of the coil wire is going to be copper, which is the obvious choice because of its low resistivity and availability in thin dimensions. The properties of copper is listed in table 3.1

Parameter	Symbol	Value	Unit
Density	ρ	$8.92 \cdot 10^3$	kg/m^3
Resistivity at $20°C$	σ_0	$1.7 \cdot 10^{-8}$	Ωm
Temperature coefficient of resistivity	α	$3.9 \cdot 10^{-3}$	$1/°C$

Table 3.1: Copper properties

Considerations	Inside placement	Outside placement
Physical protection	*Well protected*	Vulnerable to physical damage
Design modifications	Big structural and electrical design modifications necessary to fit	*Few design modifications necessary*
Space	Takes up place which could potentially be used to other components	*Probably no other components desired to be placed here*
Other	*Cleaner design*	May interfere physically with the Langmuir probes while not deployed. This is not yet designed.

Table 3.2: Considerations regarding placement of the long coils. Emphasized text is considered as the preferred argument.

3.2.2 Dimensions

For three axial control, three magnetorquers mounted orthogonal to each other is going to be implemented. The magnetorquers are also referred to as coils throughout the thesis. One coil will be placed on the short side, and two on the long sides. To get the most effective coils, we want them to follow the outer border of the sides, but of cause inside the deployment rails. There will be two different designs, one for the short side and one for the long side, from now of referred to as short coil and long coil. To find the exact space available for the coil, closely cooperation with the mechanical and electrical team was necessary.

The long coils had to be mounted close to two of the solar cells PCB cards. It was considered whether the coils should be mounted on the outside (solar cells side) or the inside.

It was decided to place the coils outside. The outside mounting disadvantages were considered as not likely to be any problem, and we did not want to do the necessary design modifications to place them inside.

The short coils is not designed in this thesis, because it is not decided how much available space there is for it.

3.2.3 Design Considerations

To find the right design parameters of the coils, we have to look at how different parameters affect the power consumption and magnetic moment. We already know that number of turns does not affect the magnetic moment, but it does affect resistance and hence the power consumption. The face area of the coil (A) is also affecting the magnetic moment and power consumption in a way that a bigger area gives a more effective coil. From this, it is clear that we want the coils as big as possible regarding power consumption and magnetic moment. If we consider the available space as constant, we only have a limited numbers of parameters left to adjust. Since we do not want a voltage regulator for the coils, the voltage is fixed. Since the temperature is not controllable, all parameters on the right side of 3.7 can be seen as fixed except for a^w.This gives us that the only way to adjust the magnetic moment is through the wire dimension a^w. Since the space available still is fixed, we are forced to reduce number of turns if we increase wire dimension. This again leads to less effective coil, and the power consumption is increasing. Actually a^w

affects the resistance in 3.5 both directly and through n.

To do the coil calculations, a spreadsheet was developed. The spreadsheet includes the fill factor for different wire dimensions provided by the supplier, with an option to scale this factor in case the supplier is a bit optimistic. The fill factor states how many wires that fits in $1cm^2$. A screenshot of the Magnetorquer Calculator Spreadsheet is shown table3.3.

3.3 Magnetorquer Production

A production method for the magnetorquers had to be developed. In cooperation with the mechanical workshop, a coil winding machine have been designed and produced. The design goals for the coil and coil winder was

1. Effectively placement of each wire turn, to achieve high turn count.
2. Mechanical stiff coil to maintain its shape.
3. Mounting to PCB must be taken into account.
4. Adhesive must be low outgassing, space qualified.
5. Coil wires must be connected to thicker cables, and the splice must be mechanical robust.
6. The coil winder must be adjustable to different coil parameters.
7. The coil winder must count turns while winding.

3.3.1 Coil Winder

The coil winder was produced as showed in figure 3.2. The coil winders functionality is a motor winding a wire on to a custom reel, in the size of the desired coil. The wire is guided on to the reel with a servo. The reel is filled with wire and adhesive, and the result is a coil bonded together as one part. Each component of the coil winder is described in the following sections.

Coil Reel

A rectangular reel with the dimensions of the desired coil is fabricated in plastic. The coil reel was first made as a Teflon reel, but before a suitable adhesive was obtained, a 3D-printed version was produced. It turned out that this method was very successful, and a magnetorquer was never produced in the Teflon reel. A description of both of them follows:

The Teflon reel is a five layer reel, where the outer two layers are made of plastic while the inner three are made of Teflon. The Teflon layers are creating a reel in itself, but needs the outer plastic layers to obtain stiffness. All the layers are mounted together with $3mm$ screws, able to disassemble when the coil is ready to be removed. The inner reel is made of Teflon because of it low friction, this makes it easier to remove from the adhesive utilized to bond the coil wires together. The resulting coil is only wire and adhesive, and the reel layers can be used to make more coils.

The 3D printed reel is a three layer reel, where the outer two layers are the same parts as in the Teflon reel. The middle layer is instead of Teflon, plastic fabricated using

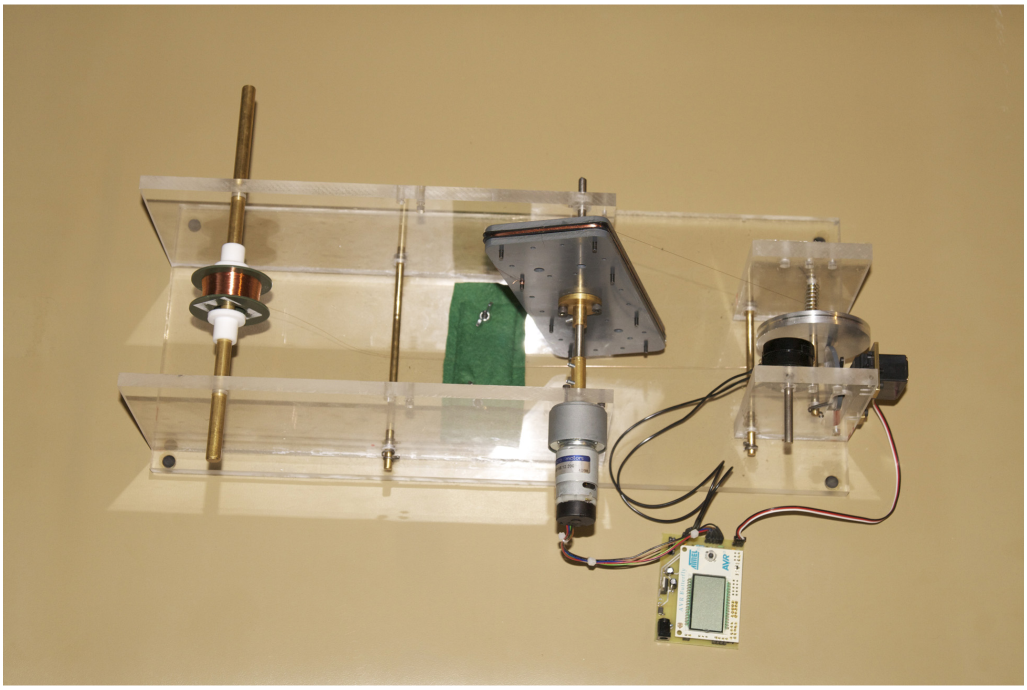


Figure 3.2: The Coil Winder, fully developed to make the CubeSTAR magnetorquer.

a 3D printer. The middle layer is made up of two parts, a core and a coil frame. The idea is that the printed coil frame will be a part of the final coil. One coil frame must be printed for each coil produced. The advantages of this method are that the frame protects the coil wire well. Mounting holes can easily be added, and design changes can easily be performed. The disadvantage is the extra space consumed.

Mechanical

A copper wire reel is placed on a shaft in the back of the coil winder. The wire is led forward through a wire break. The principle of the brake is a plastic plate which is pushed to the bottom plate of the coil winder. The wire is passing through between the two plates and protected by felt on both sides. The tension of the break is adjustable with two wing nuts adjusting the space between the plates. The wire is further passing through an aluminum guiding wheel. The guiding wheel is controlling the placement of the coil wire on the coil reel. The coil winder construction is made up of Plexiglas, metal shafts and Teflon bearings.

Motor

The motor utilized is a Micro Motors RH158-12-200 controlled by a potentiometer. The DC motor includes a hall-effect encoder and 200:1 gearing, which makes it suitable for the coil winder. The low output speed from the gearing enables the motor to be mounted directly to the coil reel shaft. The hall-effect encoder internal circuitry has an open drain MOSFET as output, and can be connected directly to a microcontroller with a suited pull up resistor.

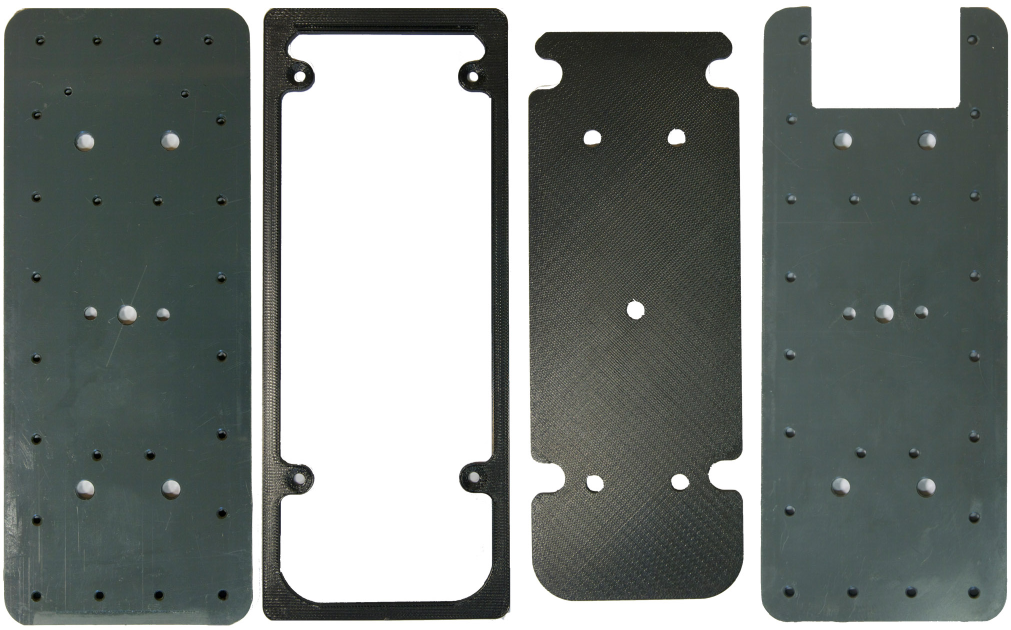


Figure 3.3: The Coil reel components. The components above is mounted together as a sandwich, with the two outer parts holding the two middle parts in between them. The two middle parts are made with a 3D-printer, fits into each other, and can easily be adjusted in the CAD model. The second part from left is the wire frame, which will be the visual part of the final magnetorquer.

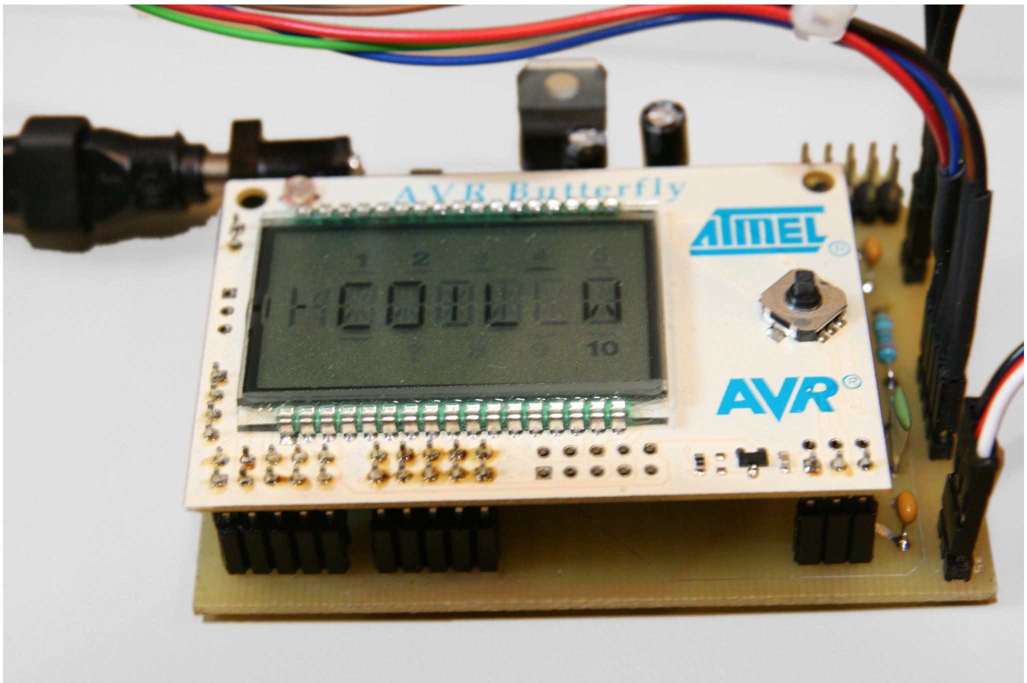


Figure 3.4: Coil winder card, with the AVR Butterfly stacked on top.

Coil Winder Card

It was crucial to the coil winder project to design the electronics as fast as possible. An Atmel AVR Butterfly was desired to be utilized, from now of referred to as the Butterfly. The Butterfly is a demonstrating kit from Atmel, including an ATmega169 microcontroller, a six characters Liquid Crystal Display (LCD) and a four-direction joystick with center push. The LCD and the joystick made it suited to make a menu system adjusting parameters on the coil winder. The Butterfly is originally a battery powered device, without any external connectors, but most of the pins from the microcontroller are routed to holes on the PCB, so pin row connectors can be added. To wire up all the electrical parts of the coil winder, an electric circuit was designed, and a PCB card produced. The PCB card is from now of referred to as the coil winder card. The coil winder card is a simple two layer card, etched at the electronic workshop. The coil winder card was designed for the Butterfly to be stacked on top of it. The pin row connectors are both electric connectivity and mechanical bearing. The coil winder card has a $12V$ input. The motor is supplied with the unregulated $12V$, while the Butterfly and the servo have a voltage regulator delivering$3.3V$. The coil winder card is interconnecting all of the components, and makes the whole setup to only require a $12V$ source.

Guiding Wheel and Servo

The magnetorquer thickness is a few millimeters, and the wire thickness is more than one magnitude less. It was desired to wind the coil as effective as possible. In one coil reel turn, the wire guider has to move one wire thickness sideways. The movement of the guiding wheel must be controlled with accuracy at approximately $0.1mm$. A Parallax

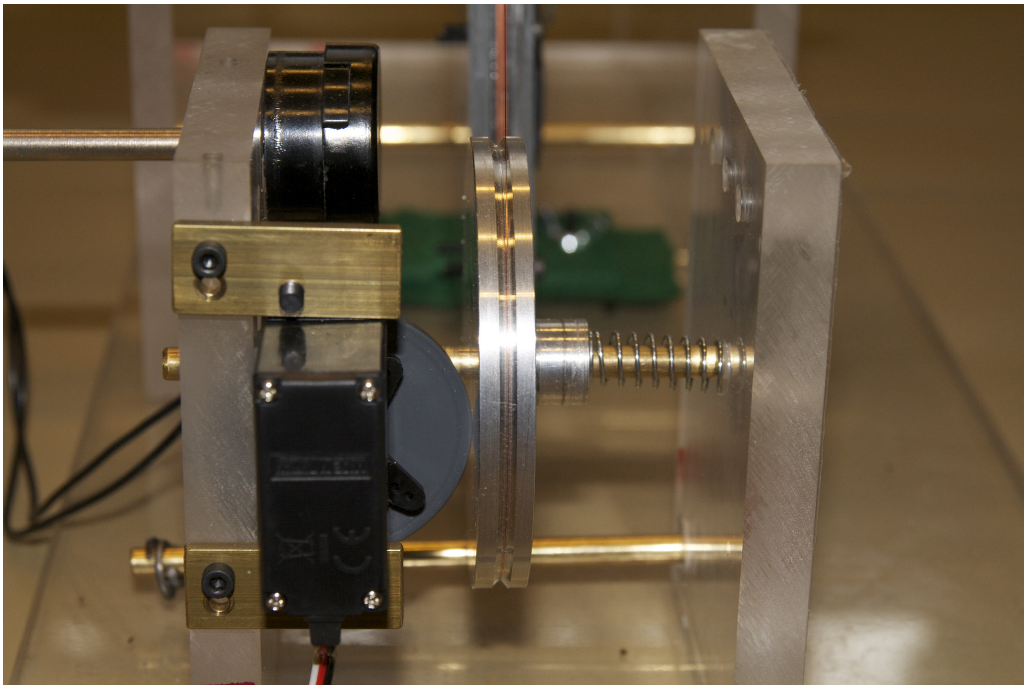


Figure 3.5: The wire guiding part of the Coil Winder. In this picture, we can see the servo with the plastic dish pushing at the guiding wheel. In the upper left, the potentiometer controlling the motor is pictured.

900-00005 servo was utilized. The servo has a 180° range, and is controlled by a Pulse With Modulated (PWM) signal (see section 4.2.2). To achieve the best possible accuracy of the guidance it is important that a big part of the servo range is utilized to move the wheel a short distance, just slightly longer than the thickness of the coil. A gearing functionality was achieved by mounting a circular plastic dish off center on the servos rotation axis. When rotating, the dish is pushing at the guiding wheel. A spring is making the guiding wheel follow the dish all the time. If d is the distance from the rotation axis of the dish to the edge touching the guiding wheel, and the dish is mounted so that d is at its lowest value at rotation $\alpha = 0°$,

$$d = r + o \cdot cos(\alpha)$$

Where r is the radius of the dish and o is the offset between center of the dish and the rotation axis.

A servo is built up of a DC motor, a potentiometer, gearing and control circuitry. The PWM signal controlling the servo is a pulse once every $20ms$ with a duration of $1ms$ to $2ms$. The length of the pulses represents the desired rotation, from correspondingly 0° to 180° rotation. Additional to the PWM signal, the servo requires power, in total three wires. The servo was chosen as actuator just because of its simple control. Alternatively a stepper motor or a motor with an encoder could have been used, but would have required a more complex design.

Coil Winder Microcontroller Code

The AVR Butterfly is well suited for menu system navigation. The firmware developed for the microcontroller includes a menu system based on a Finite-state machine. The functions implemented is

- Setup
  - Set with of coil between $0 - 4mm$
  - Set wire cross-sectional dimension
  - Set the outer positions of the servo range
  - Reset counter
- Run
  - Turn counter on the display
  - Servo controller with cosine correction
  - Override possibility

The firmware was developed as described in section 4.4.1, except that the microcontroller was not a XMEGA. The menu system and state machine is based upon Atmel AVR Butterfly Rev07 application, but have been rewritten to fit this purpose and the avr-gcc compiler. A library special written for the LCD display on the AVR butterfly was also utilized, the library is written by Dean Camera and freely published on the Internet. A fully overview of the firmwares functionality and use are given in table A.1 in appendix A.

3.4 Design Results and Future Work

It is produced one magnetorquer containing 269 turns of $0.15mm$ copper wire. It was measured to a self inductance of $33.8mH$, with 149.5Ω resistance at room temperature.A user manual describing the whole process of making a coil is available in appendix A. In figure 3.3, a new coil frame is produced, with mounting ears. The coils are planned to be placed upon two of the solar cell PCB cards, surrounding the solar cells. It should however be considered to make the PCB fit inside the main part of the coil frame, like in figure 3.6. This will make the coil frame's upper side to be flush with the structural corner beams. The coil will then be unexposed and well protected. It is however very important that there is proper isolation between the wires and the conducting aluminum. It may be necessary to reduce the coil turn some. A short side coil is not considered in this thesis, since the place available is not able to determine at this time. By using the same methods as developed for the long side coil, it should be fairly easy to produce a short side coil.

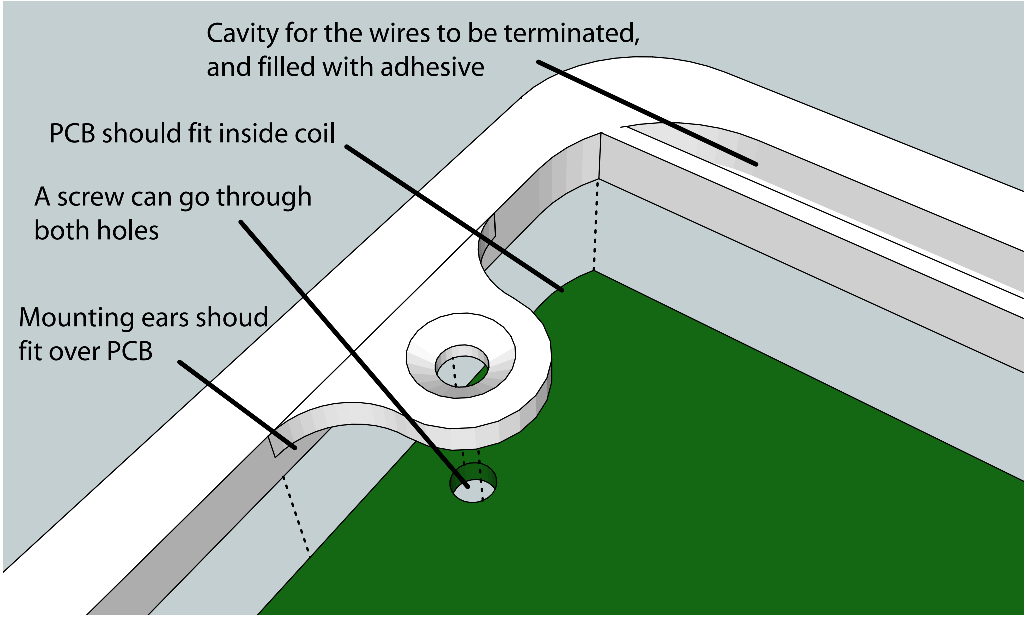


Figure 3.6: A picture describing the improvements performed or planned, compared to the magnetorquer produced, pictured in figure 3.7.

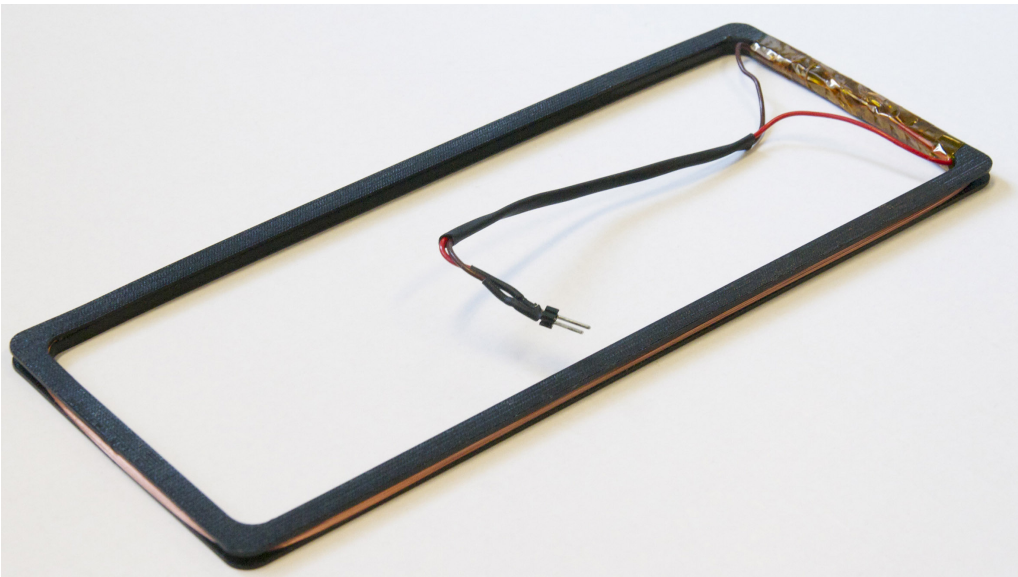


Figure 3.7: The first magnetorquer produced. This version does not have inner mounting ears, as the frame in figure 3.3.

Magnetorquer Calculator

Constraints

Maximum width	b max	82	mm
Maximum height	h max	240	mm
Maximum coil cross-sectional width	b_c max	2	mm
Maximum coil cross-sectional height	h_c max	5	mm
Voltage at full load	U_c	3,6	V
Maximum allowed current	I_max	100	mA
Minimum temperature	T_min	-60	C°
Nominal temperature	T_norm	25	C°
Maximum temperature	T_max	100	C°

Copper properties

Density	ρ	8,92E+03	kg/m3
Resistivity at 20 degrees	σ_0	1,68E-08	Ωm
Temperature coefficient of resistivity	α	3,90E-03	1/C°

Calculated sizes

Maximum face area	a_s max	19680	mm2
Maximum coil cross-sectional area	a_c max	10	mm2
Mean width	b mean	80	mm
Mean height	h mean	238	mm
Mean face area	a mean	19040	mm2
Mean circumference	l mean	636	mm

Choosed values

Wire diameter	d_w	0,15	mm
Manually inserted turns		269	turns

Coil fill estimation

Scale fill factor		0,75	
Mass	M_s	27,67	g
Wire cross sectional area	a_w	1,77E-02	mm2
Coil cross sectional area	a_s c	4,88	mm2
Fill factor		2764	Wires/cm^2
Calculated turns	N_s	276	turns

Calculated coil properties

Turns used in furthure calculations		269	turns
Mass	M_s	26,97	g
Wire cross sectional area	a_w	1,77E-02	mm2
Coil cross sectional area	a_s c	4,75	mm2
Fill factor		3686	Wires/cm^2

Calculations for above stated specifications		min	nom	max	unit
Resistivity (σ)	σ	1,16E-08	1,71E-08	2,20E-08	Ωm
Resistance	Ω	111,90	165,82	213,39	Ω
Maximum Current at	I_s	32,17	21,71	16,87	mA
Maximum Power at	P_s	115,82	78,16	60,73	mW
Produceable magnetic moment	m_s	164,77	111,20	86,41	mAm^2
Produced magnetic moment per current		5,12176	5,12176	5,12176	m^2

Table 3.3: A Magnetorquer Calculator Spreadsheet was developed, to easily calculate desired magnetorquer parameters. The values in this table are the resulting size, wire diameter and turn count of the produced magnetorquer, and calculated properties.

Chapter 4

Electronic Design

This chapter is presenting the design process of the *ADCS* card and the *mini-backplane* card. Despite the name of the ADCS card, it was never intended to do any determination during this thesis, but is the first parts of the complete ADCS system. The rest of the ADCS components are planned to be added to this card in later revisions.

4.1 Electronics on the CubeSTAR

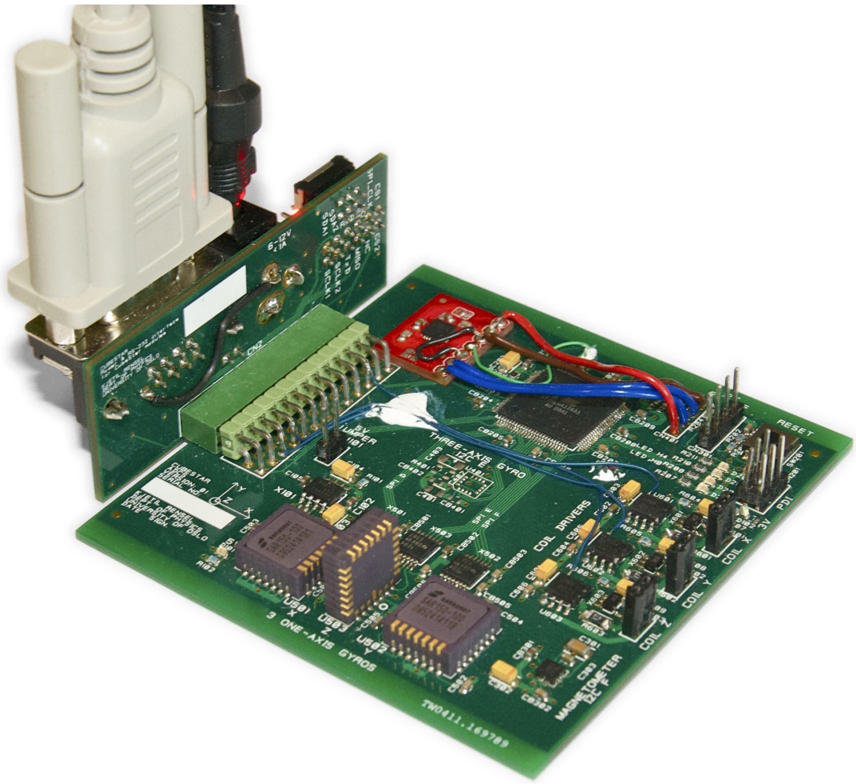
The electronics made to the CubeSTAR have to be designed so it fits into what is already developed on the project. Both electrically and mechanical, the ADS card have to comply with the module requirements for CubeSTAR. CubeSTAR has a backplane placed at one of the long sides of the satellite, which is connecting module cards, solar cells cards and the battery pack together. For each module slot, the backplane has a 26-pin connector. The backplane is designed by the Electronic workshop at the institute, and is at this time at engineering version 1. The main functionality of the card is:

- Interconnection between module cards, solar cell cards and the battery pack
- Provide and control power to each module card
- Communicate and receive information of each module card. (On Board Data Handling)

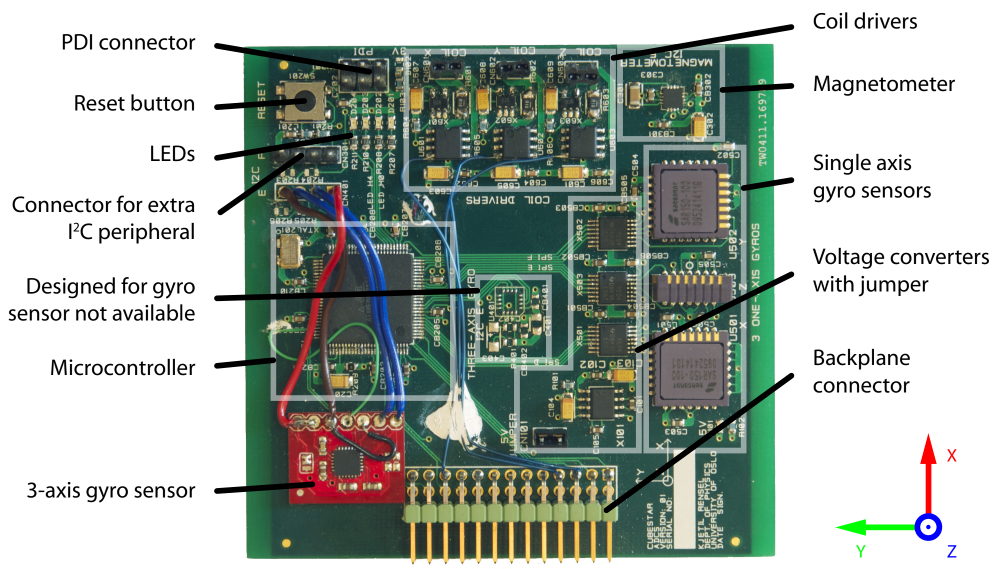
Each slot has a corresponding mechanical slit in the structure which fits the thickness of a PCB card. The slit is the main bearing for the card. The ADCS card is going to be located in one of the module slots, and have to comply with the specifications. Refer to B for drawing of the card, keep attention to the keep out areas at each side, which is where the card is going to fit into the slits.

The backplane connector contains several communication lines. It is not fully decided how those are going to be utilized by the different module cards. To be sure the ADS card is designed to support future implementations by connecting all lines to the microcontroller. Since the on board data handling does not have been implemented yet, only the UART communication lines are utilized in this thesis.

The power provided to the module cards is $3V$, able to deliver a current up to $1A$. Current sensors on the backplane will disconnect power if this limit is exceeded. Since all of the defined pins on the backplane connector are utilized by the ADCS card, the pinout for the backplane connector is equal to the pinout found in the ADCS card schematic.



(a) The ADCS card connected to the Mini Backplane card



(b) Description of the logical blocks, with its defined directions in the bottom left.

Figure 4.1: The ADS card. Following the module card specification to fit in the satellite.

4.2 Hardware System Architecture

The ADCS card is mainly designed to be able to detumble the satellite without any other requirements than power. We decided early to implement a gyro, even if this is not strictly necessary to perform detumbling. However is it desired for the rest of the ADCS. The hardware is designed to fulfill the following tasks:

- Measure magnetic field
- Measure angular velocity
- Control three magnetorquers
- Perform a detumbling algorithm
- Deliver measurement data to other sub systems
- Give control of the magnetorquer to other sub systems
- Communicate with a computer for testing and verification

An Atmel ATXMEGA128A1 microcontroller is implemented as interfacing between sensors, magnetorquer control and the communication lines on the backplane. Pre-processing of sensor data and detumbling are also being implemented into the same microcontroller. A gyro and magnetometer is implemented using small form factor consumer sensors. The components and its circuitry is explained in detail in the next sections

4.2.1 Microcontroller Circuitry

The Atmel AVR ATXMEGA128A1 microcontroller was implemented as the interfacing and controlling unit of the ADS card. The ATXMEGA128A1 is a low power microcontroller with a variety of internal peripherals. The unit has a 100-pin package, able to connect the backplane and the different sensors on separate communication buses. ATXMEGA128A1 units are also utilized on the backplane card and on the communication card. Those properties made it an easy choice to utilize this specific model.

The ATXMEGA128A1 pins are divided into 11 different *ports.* A port is a collection of pins on the unit, where 9 of the ports consist of 8 pins. While all of the pins which are member of a port can be used as general input or output, most pins also can have special functions like interconnection bus,PWM and analog interfacing. On the ADS card, the microcontroller is wired to communicate to several on-board ICs, additionally to the backplane. The communication is using several different bus standards, and as far as possible, a separate bus is utilized for each unit. Separate buses is used to avoid an eventual dysfunctional unit to block the communication line for functional units. In the case of too few pins or peripherals on the microcontroller in later revisions, several units can be connected to the same bus. In table 4.1, an overview of which purpose each of the pins contains. The table displays what is connected after some modifications compared to the schematic. The table complies to the source code.

4.2.1.1 XMEGA Clock System

According to the application note [29], the ATXMEGA128A1 contains several internal clocks utilized by the system processor and many of the internal peripherals, like timer/counters and communication interfaces. It is possible to run the microcontroller

PORT \ PIN	0	1	2	3	4	5	6	7
A	BP Connector		ITG3200					
	CS1	CS2	Data Ready					
B	Current monitor							
	X coil	Y coil	Z coil					
C	BP Connector I2C-1		BP Connector UART		H-Bridge PWM		PB Connector SPI	
	SDA1	SCLK1	UART RxD	UART TxD	Forward Z	Reverse Z	MISO	SPI CLK
D	H-Bridge PWM				X-axis SAR150			
	Forward X	Forward Y	Reverse X	Reverse Y	LOAD	MOSI	MISO	SPI CLK
E	ITG3200-I2C				Y-axis SAR150			
	SDA	SCLK			LOAD	MOSI	MISO	SPI CLK
F	HMC5883 - I2C				Z-axis SAR150			
	SDA	SCLK	Data Ready		LOAD	MOSI	MISO	SPI CLK
H	LED0	LED1	LED2	LED3				

Table 4.1: XMEGA port description, as they are utilized on the modified ADS card. Grayed out text is not implemented in source code.

completely on any of these but one. An external clock is still implemented, to make sure a reliable and stable clock source is present, and to freely choose the clock frequency. The microcontroller's power consumption is strongly dependent of the clock speed. The clock stability is also crucial to maintain the right timing on UART communication. In this thesis, a $3.6864 Mhz$ crystal oscillator is utilized. It is a fairly low speed, which also is fits many standard UART baud rates.

4.2.2 Inter Communication

The different communication standards utilized on the ADCS project is briefly explained.

I^{2}C

Inter-Integrated Circuit (I^{2}C) is a bus standard able to connect several units together by only two wires. I^{2}C is also referred to as Two-Wire Interface (TWI) among other by Atmel. Electrically, the two lines Serial Data Line (SDA) and Serial Clock (SCL) are pulled high by resistors. For devices to communicate, a master must initiate the transaction, and supply clock to the SCL. Every slave has its own address, and only utilizes the line when requested by the master.

SPI

Serial Peripheral Interface Bus (SPI) is a synchronous bidirectional serial bus. SPI utilizes four wires to operate between two devices. Data is always sent both ways at the same time on two separate wires, while *clock* and *Slave Select* is used to respectively clock the communication and enable the slave. Several slaves can be connected to the bus by either daisy-chain or parallel connect the data lines. The slave select line is shared when daisy-chained, but must be separated to enable one slave at a time when connected in parallel.

UART

Universal Asynchronous Receiver/Transmitter is a hardware peripheral able to serially send and receive data. The transmission lines connecting UART devices together do also commonly share the same name. A serial link is asynchronous when a clock is not present as a separate signal, like on standards such as SPI and I^2C. A byte transmitted over an UART link starts and some times ends with bits defining start and/or end of a byte. An UART link varies in structure, and several options must be chosen for two units to communicate. Options include transmission speed (baud rate), stop bits, and error correction bits. The microcontroller utilized in this thesis has a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) port, which makes it possible to configure it for a synchronous data link. When a UART communication is desired between electronic devices, the logical signal is normally converted to a standard defining electrical property. Standards include RS-232, RS-422 or EIA-485. A RS-232 port is present in most desktop PCs, known as COM-port in Microsoft operating systems.

In this thesis, a UART baud rate of 115200 is utilized, which is a fairly high speed for a RS-232 link. The relatively high speed is chosen, so that the microcontroller and the PC client software do not have to wait a needlessly long time for the data to be sent. A higher sample rate is also achievable due to the baud rate. The baud rate calculation is performed by the spreadsheet included in [33].

PDI

Program and Debug Interface (PDI) is an Atmel proprietary protocol for programming and debugging of devices like the ATXMEGA MCU. In the physical layer, the PDI uses a half-duplex USART, containing one line for data and one for clock[24]. In this thesis, the PDI is utilized in favor of JTAG, which would fulfill the same tasks with the same equipment. The PDI was chosen mainly because the connector has a smaller footprint than the JTAG connector.

PWM

Pulse-with modulation (PWM) is a technique for controlling or representing an analog value. The value is only represented by the duration (with) of the signal fully on and fully off. The PWM signal is used in a wide variety of applications, and is in this thesis utilized on the ADCS card and on the coil winder card. On the two cards, PWM is utilized in different ways, explained in the corresponding sections.

4.2.3 Sensonor SAR150 Gyro Circuitry

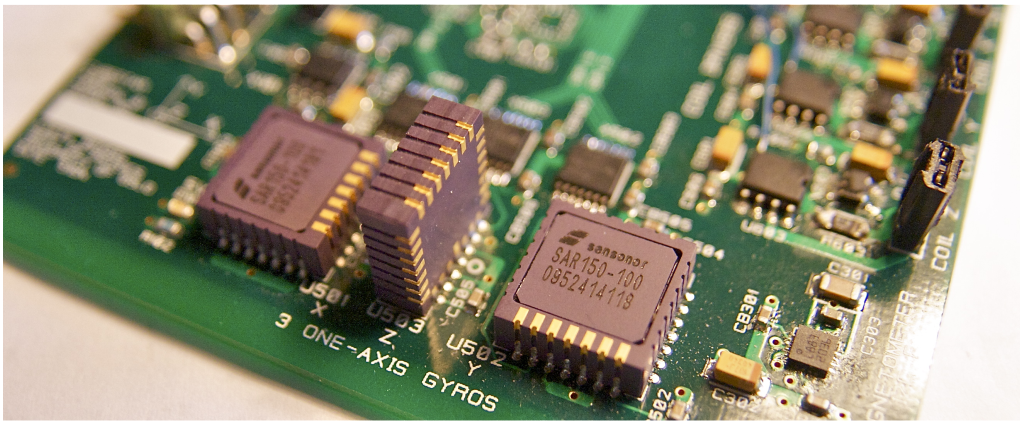
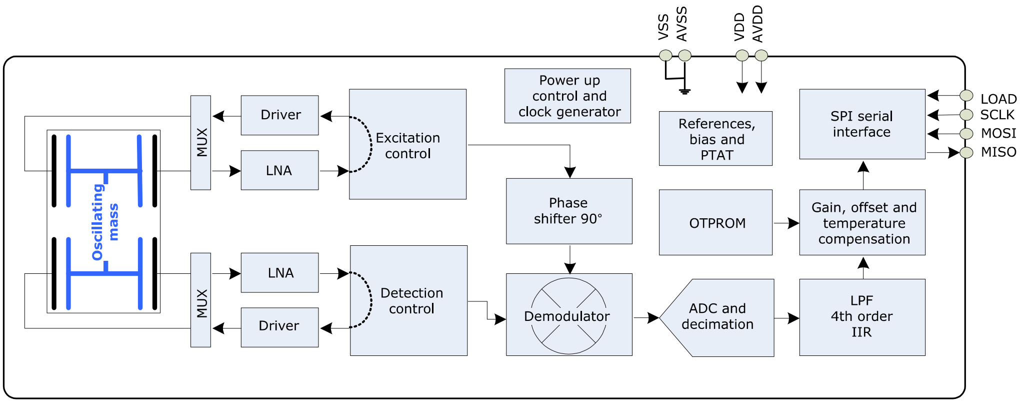


Figure 4.2: Three SAR150 mounted in a three axis setup on the ADS card. The SAR package is designed to be mounted both in normal and upward position. In the lower right corner it is possible to see the HMC5883 magnetometer, which is significantly smaller.

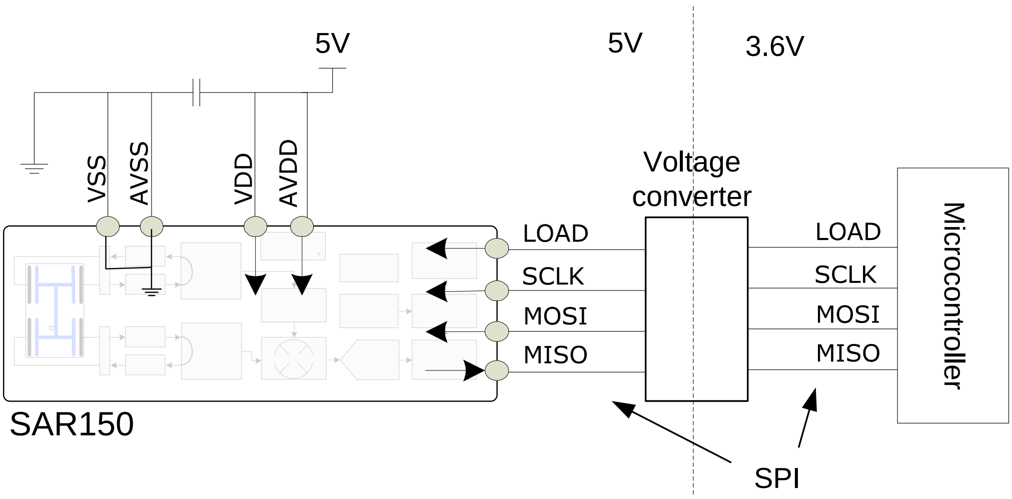
In an early stage of the project, it was desired to test a MEMS Gyro sensor from Sensonor. In [25], a master thesis written at the Institute, MEMS Gyro sensors from Sensonor is tested. We contacted Sensonor, which supplied us with three SAR150-100 high precision sensors for free. The SAR150-100 is a one axis high precision MEMS gyro sensor with ASIC providing digital circuitry including Serial Peripheral Interface (SPI). The SAR sensors are coming in two models, with several different input ranges for each model. The two models are named SAR100 and SAR150, where SAR150 is the high precision unit of the two. The number after the dash, indicates the input range in unit of $^{\circ}/s$. The SAR sensor has a Leadless Ceramic Carrier (LCC) package containing terminals on two perpendicular sides. This enables the chip to be mounted in both horizontal and vertical position, and unlike most one-axis gyros on the market, these feature makes three identical sensors able to measure all axes, even if they are mounted on the same PCB.

The SAR sensor is built for $5V$operation, while our system is $3V$. A $3V$ to $5V$ voltage pump regulator was applied to provide the SAR sensors with correct voltage. MAX682ESA from Maxim was chosen due to its availability and ease of use. The regulator is fixed at $3V$ to $5V$ regulation and does not require any loopback or reference voltages. In addition to the regulator, level voltage shifter for the SPI lines was necessary. A TXB0104 bidirectional voltage-level translator for each sensor was chosen. This chip works as a signal bridge between the two voltage domains. The TXB0104 is present on Sensonor's own evaluation board for the SAR sensor, which made it a safe choice. The TXB0104 is also easy to implement, with no extra circuitry except bypass capacitors.

The devices communicate to the microcontroller via SPI. Each of the sensors is connected to separate SPI ports on the microcontroller, even if daisy chaining is possible. It is designed this way, to guarantee that a malfunction in one device would not block the communication line to the rest of the sensors.



(a) SAR150 internal block schematic. [26]



(b) SAR150 external circuit. Based on [26]

Figure 4.3: SAR150 Schematics

4.2.4 3-Axis Single Chip Gyro Sensor

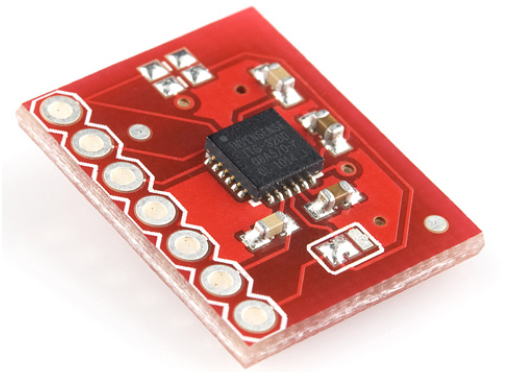


Figure 4.4: InvenSense ITG-3200 3-axis gyro sensor mounted on a breakout board.

Three axis gyro sensors have been cheaper and more easily available lately, due implementation in consumer products like gaming consoles and smart phones. We wanted to compare the high precision sensors from Sensonor with one of those. When the ADS card was designed, it was decided to test STMicroelectronics L3G44200D. Units was ordered, and designed to fit on the ADS card. Unfortunately, the supplier was unable to deliver the L3G44200D, and another solution had to be made. An InvenSense ITG-3200 sensor mounted on a breakout card was ordered. Necessary test could still be performed without any delays. The breakout card is med by SparkFun Electronics, which also was its supplier. The ADS card was already designed with the ability to connect separate I^2C units. The sensor breakout card was taped to the ADS card by double-sided tape, and power, I^2C and data ready was soldered on. In figure 4.5, a block schematic of the sensor is illustrated. The unit is fully controlled by setting values in the Config Register through I^2C.

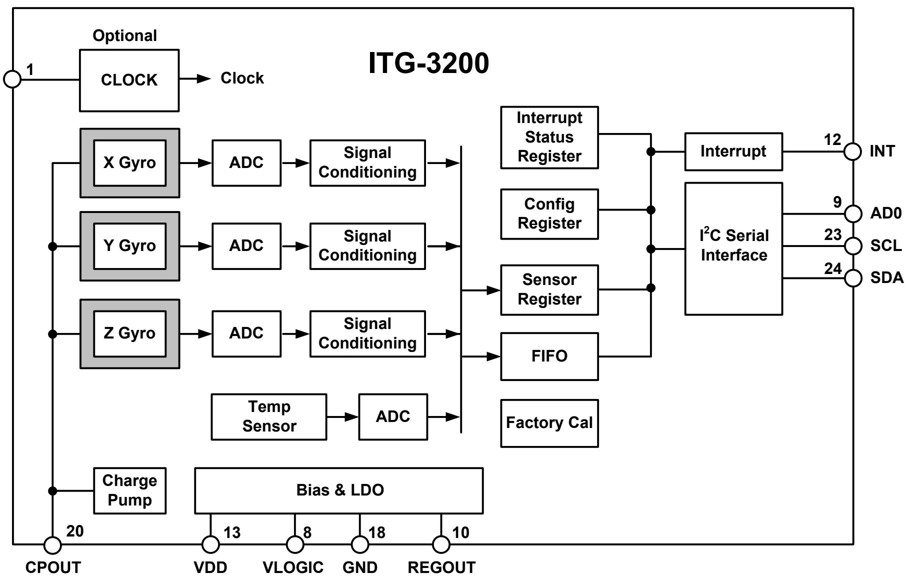


Figure 4.5: ITG-3200 internal block schematic. [27]

4.2.5 Magnetometer Circuitry

Honeywell is one of the market leaders producing magnetic sensors. In the design phase of the ADS card, The HMC5843 3-axis magnetoresistive sensor from Honeywell was tested on an evaluation board. The HMC5843 got a successor right before the ADS card was produced, and we decided to implement the new HMC5883L instead. The HMC5883L is slightly better at most parameters, and is a safer choice regarding availability. The HMC5883L comes in a $3 \times 3 \times 0.9mm$ 16-pins LCC package, and requires only additional capacitors to work properly. The sensor communicates via I^2C line acting as a slave. The I^2C lines and data ready interrupt output pin (DRDY) are connected to the microcontroller. DRDY enables activation of an interrupt on the microcontroller when sample data is ready to read. Use of a data ready line prevents the microcontroller from waiting and polling on the magnetometer. In 4.6 the internal functions of the sensor is presented together with its external requirements. Internally, the device has two different power domains, one for the internal functions, VDD, and one for IO interface, VDDIO. VDDIO accepts lover voltage than VDD, allowing the chip to communicate with low voltage external units. We do not need this feature, and connects both of the voltage inputs together. Ironically, if this feature had been present on the SAR150 gyroscope, we would not have needed the bidirectional voltage-level translators on the SAR150 SPI bus.

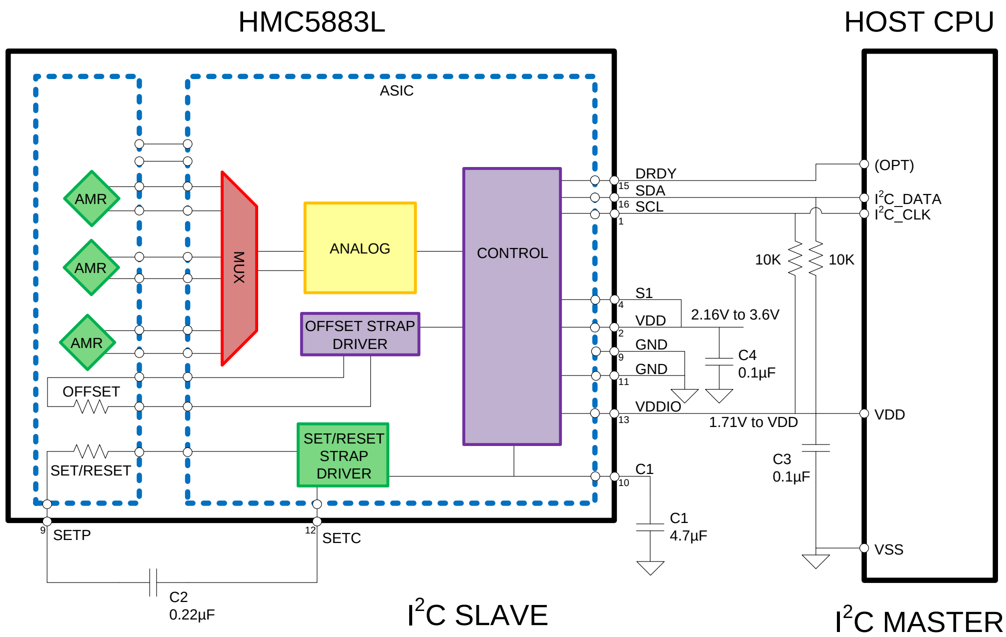


Figure 4.6: Schematic diagram of HMC5883L and its necessary circuitry. [28]

4.2.6 Magnetorquer Driver H-bridge

It is necessary to control the direction and amount of current flowing through the magnetorquers. Control circuitry is implemented on the ADCS card by utilizing three H-bridge ICs. An H-bridge consists of four transistors, two transistors connected to each output.

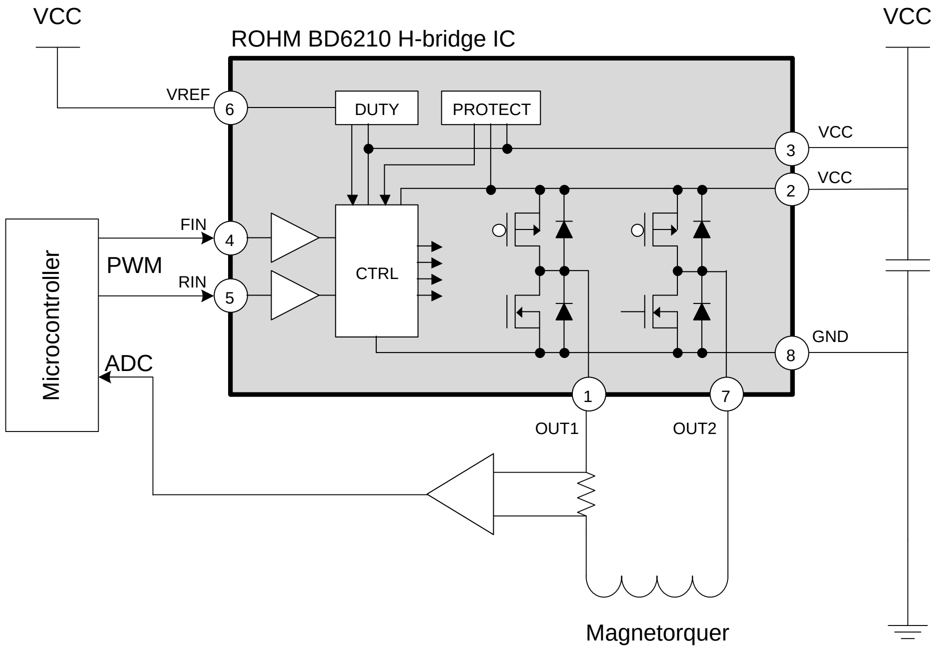


Figure 4.7: Schematic diagram of ROHM Semiconductor BD6210. Based on[30]

One of the two transistors connected to an output is connected to VCC, and the other one to GND. In that way, an H-bridge is able to connect both outputs individually to either VCC or GND, and thus control the direction. The amount of current is regulated by the PWM principle, fast switching on and off. H-bridges are commonly used circuits to control motors, and many IC versions are available. The BD6210F from ROHM Semiconductor was implemented on the ADCS card. The BD6210F is a single channel H-bridge with maximum current output at $0.45A$. In 4.7 a schematic view of the internal functions of the BD6210F is shown together with its external logical circuit. The H-bridge is built up of MOSFET transistors controlled by the control unit. In 4.2 the various control states are listed. The unit can additionally to the PWM also be controlled with an analog signal on the VREF input, this is not further discussed. As indicated in table 4.2, the PWM signal maintains its timing on the output port, but the signal is inverted.

4.2.6.1 Controlling the Magnetorquer

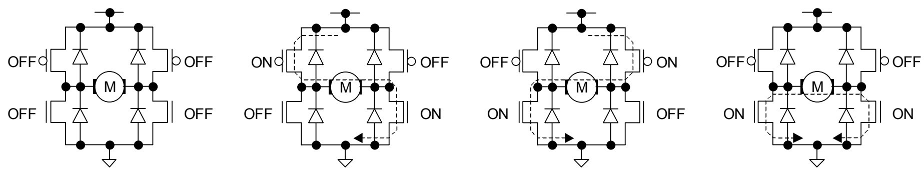
The magnetorquers are inductors including all its inherent properties. They will try to resist changes in current, and some care must be taken. In inductor with a current running constantly through it, energy is stored as a magnetic field in the coil. When the external source is turned off, the magnetic field will induce a back emf (electromotive force) trying to maintain the current flow. In a situation where the inductor is disconnected when the magnetic field is present, the induced current does not have anywhere to go, and high voltages can be present. In the H-bridge IC, diodes are placed in parallel with each transistor. In that way, the current has a way to pass even if all transistors are switched off. Theoretically, the current will go back the power source path, and charge a capacitor or eventually the battery.

In figure 4.8, the four basic modes of the H-bridge are illustrated. Mode *d,* Brake mode, are utilized to break a motor in rotation. The back emf generated is opposite the direction which is needed for the motor to run forward, and hence it is breaking. For our purpose, we want the magnetorquer to be in any of the three former modes. In table 4.2, we can see that mode *e* to *j*, lets you alter between two of the four *basic* modes *a, b, c* and *d.* The modes we want to utilize are *e* and *f* when running, and *d* when idle.

	FIN	RIN	VREF	OUT1	OUT2	Operation
a	L	L	X	Hi-Z*	Hi-Z*	Stand-by (idling)
b	H	L	VCC	H	L	Forward (OUT1 > OUT2)
c	L	H	VCC	L	H	Reverse (OUT1 < OUT2)
d	H	H	X	L	L	Brake (stop)
e	PWM	L	VCC	H	$\overline{\text{PWM}}$	Forward (PWM control mode A)
f	L	PWM	VCC	$\overline{\text{PWM}}$	H	Reverse (PWM control mode A)
g	H	PWM	VCC	$\overline{\text{PWM}}$	L	Forward (PWM control mode B)
h	PWM	H	VCC	L	$\overline{\text{PWM}}$	Reverse (PWM control mode B)
i	H	L	Option	H	$\overline{\text{PWM}}$	Forward (VREF control)
j	L	H	Option	$\overline{\text{PWM}}$	H	Reverse (VREF control)

* Hi-Z is the off state of all output transistors. Please note that this is the state of the connected diodes, which differs from that of the mechanical relay.
X : Don't care

Table 4.2: Input and output in the different operation modes. [30]



(a) Stand-by mode (b) Forward mode (c) Reverse mode (d) Brake mode

Figure 4.8: The four basic modes of the H-bridge. [30]

4.2.7 Magnetorquer Current Sensing

4.2.7.1 Temperature Dependency

The magnetic moment generated in the magnetorquers is proportional to the current flowing through them. The current is proportional to the resistance, and the resistance varies with the temperature of the coil. In [23], a temperature analysis of the Cube-Sat CP3 Satellite was performed. Temperatures in the range of$-27°C$ to$+73°C$ was observed. These observations are not reported as extreme, and temperature differences of approximately $120°C$ must be expected. The Copper temperature coefficient α_{Cu} of electrical resistivity ρ is 0.0039$^1/_K$. The resistivity after a temperature change is given by

$$\rho = \rho_0(\alpha\Delta T + 1)$$

The scale factor of a $120°C$ change is then given by

$$\alpha\Delta T + 1 = 0.0039\,{}^{1}/_{\mathrm{K}} * 120°C + 1 = 1.468 \tag{4.1}$$

The resistance R is proportional to the resistivity given by

$$R = \rho\frac{l}{A}$$

,where l is the length, and A is the cross sectional area of a wire. As a result of this proportionality, the factor given in 4.1 is also applicable as a resistance scale factor in the same temperature range.

4.2.7.2 Implemented Current Measurement

To regulate the magnetic moment unaffected by the big differences in the resistance, we implemented a current sensor in the magnetorquer loop. A low resistance resistor (shunt resistor) is inserted between the magnetorquer and the H-bridge. The differential voltage across the resistor is amplified, and measured with the microcontroller's analog to digital converter (ADC). A Texas Instruments INA138 Current Shunt Monitor was utilized as the amplifier. The INA138 IC has an internal op-amp and a transistor which converts the differential input voltage to a current on the output of the chip. An external resistor R_L converts the current back to a voltage, with a gain proportional to R_L. The voltage is given by

$$V_{out} = I_S R_s R_L \cdot G \tag{4.2}$$

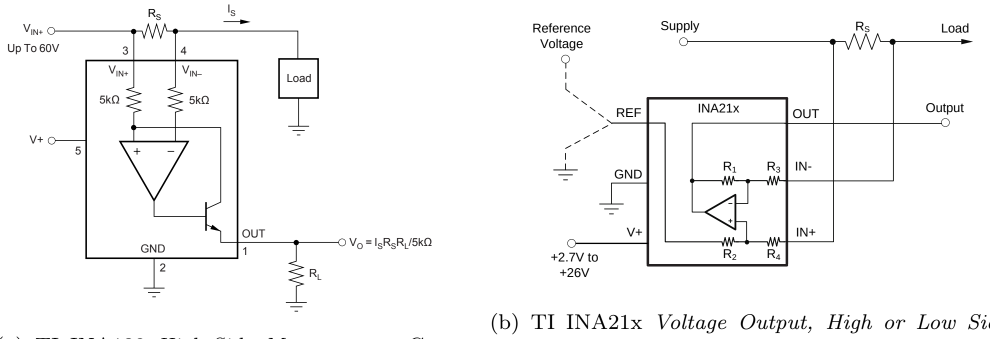
where I_s is current through the shunt resistor with the resistance R_s. With the chosen values inserted, included an estimated max value for I_S, we get

$$V_{out_{MAX}} = 40mA \cdot 0.2\Omega \cdot 604K\Omega \cdot 200\,{}^{\mu\mathrm{A}}/_{\mathrm{V}} = 0.97V$$

The INA138 is also utilized on the backplane card, and was chosen on the ADCS card to keep a coherent design. At the time chosen, we did not realize that the INA138 is a *high-side* monitor, only functional when placed on the VCC side in the current loop (high side). Since the H-bridge also connects the sensor to the low side in the loop, the INA138 is not a suited shunt monitor when placed on the output of the H-bridge. This has to be corrected for the next revision of the ADCS card. In the following sections, considerations of how this can be done are presented.

4.2.7.3 Differential Amplifier Improved Solution

A common solution is to measure the current through an H-bridge, is to place the shunt resistor outside of the H-bridge. If the shunt resistor is implemented on the low side of the driver IC, it will cause the IC not to be directly connected to GND. This is not a good practice, and should be avoided. An additional bi-effects of placing the shunt resistor outside the driver IC, is an error corresponding to the driver IC's total power consumption and the inability to measure the direction of the current. A better solution is to implement a current sensing circuitry where we already have placed the shunt resistor, on the output of the H-bridge. A differential amplifier circuit is easy to design of discrete components, but even easier with one of the many compact IC versions. In figure 4.9b, a logic schematic of an suited INA21x series monitor is illustrated. The internal op-amp and its four surrounding resistors create the classic differential amplifier. In these



(a) TI INA138 *High-Side Measurement* Current Shunt Monitor

(b) TI INA21x *Voltage Output, High or Low Side Measurement, Bi-Directional Zerø-Drift Series* Current Shunt Monitor

Figure 4.9: Schematic drawings of two Texas Instruments Current Shunt Monitors. The design of the INA138 is not suitable when connected on the low side of a current loop. However, a differential amplifier is well suited, here represented by schematic of an INA21x.

units, $R_1 = R_2$ and $R_3 = R_4$, we define these pairs respectively $R_1 = R_2 = R_A$ and $R_3 = R_4 = R_B$. The output V_{OUT} is then given by

$$V_{OUT} = (IN_+ - IN_-)\,\frac{R_A}{R_B} + V_{REF}$$

Since the sensor cannot output negative voltage values, a reference voltage has to be applied on the V_{REF}, to fulfill its purpose of amplify also negative values. A negative value on the input will now be converted to a output value between V_{REF} and GND.

4.2.7.4 XMEGA ADC

The internal Analog to Digital Converter in the ATXMEGA128A1 is a 12-bit differential ADC, in principle able to measure the differential signal from the shunt resistor directly, it even has an option of a 64x internal gain. Unfortunately, this does not work as long as the shunt resistor is placed high-side. The maximum input voltage to the ADC is V_{RefAD}, where V_{RefAD} maximum value is VCC$/1.6V$. In figure 4.10, a setup for a bi-directional current sensing is suggested. A differential amplifier circuit offsets and amplifies the voltage over the shunt resistor. The reference voltage of the differential amplifier is connected to the negative input of the ADC. The gain of the amplifier circuit, the internal ADC reference voltage and the shunt resistor must be fitted to each other, in order to get the best accuracy. Since we have to provide a reference voltage according to figure 4.10, it should be considered to also feed the ADC with its reference V_{RefAD}, which have to be at least the double of the differential amplifier reference. We choose a $1.5V$ reference voltage to the ADC, and hence a reference voltage of $0.75V$ to the differential amplifier and the negative input of the ADC. We use a similar version of equation 4.2, but now without the external resistor, and with an offset part.

$$V_{out} = I_S R_s \cdot G + V_{REF}$$

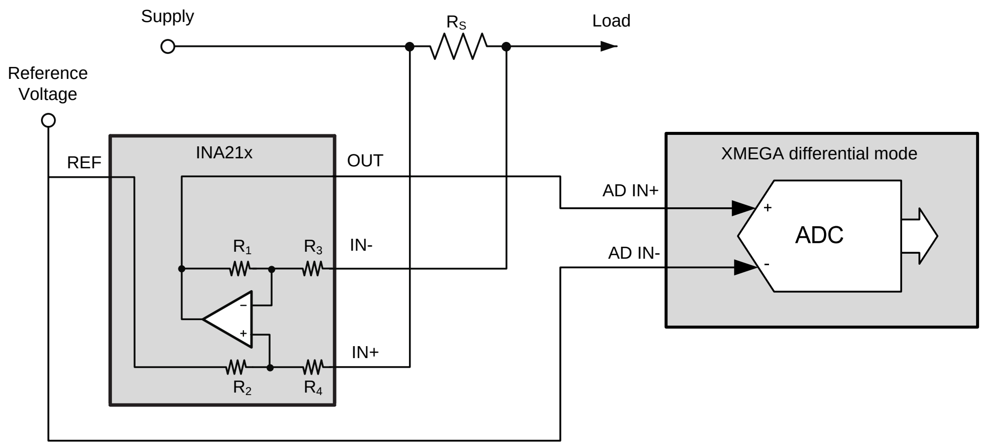


Figure 4.10: An illustration of the internal and external circuitry of an INA21x series shunt monitor connected to an Atmel XMEGA microcontroller's ADC, configured to single ended unsigned sampling.

A desired maximum V_{out} is close to $1.5V$, and the following values is proposed

$$V_{outMax} = 0.04mA \cdot 0.1\Omega \cdot 200\,{}^{\mathrm{V}}\!/\mathrm{v} + 0.75V$$

A gain value available in the INA21x series was chosen, the gain of INA210 is $200\,{}^{\mathrm{V}}\!/\mathrm{v}$. Unlike on the INA168 circuitry, there are no external components to adjust the gain.

4.2.8 PCB Design

The ADCS card is made on a four layer Printed Circuit Board (PCB). The board is made of FR-4 glass reinforced epoxy, copper lanes and gold plated connectors. The four electrical layers are from top to bottom, *top signal layer, ground layer, power layer* and *bottom signal layer.* Components are only mounted on the top side. The PCB is fully designed using the software Zuken CADSTAR and manufactured by Elprint AS. The component assembly is performed by the author at the electronics workshop.

Ground Plane

The ground plane is covering the whole area of the card in the ground layer. A ground plane design is a common way to provide a low-noise, stable ground reference throughout the card. Since all of the electrical circuitry is sharing the ground as a common $0V$ reference, it is important to have uniform ground. The ground plane design always provides a low-impedance path for return current, which is important since the ground noise will increase proportionally with the impedance. Digital circuits are a major source to noise, with its fast switching which also can occur clocked. At the same time, analog circuits are the most vulnerable to noise. The circuitry should be designed to affect the ground as little as possible, for example by adding decoupling capacitors, and minimizing the loop area of the return path. In [31], it is stated that the returning current path from digital and analog circuitry should not be crossed or shared. Several methods to split up the ground plane are presented, additional to a method which maintains the ground plane. Separation of analog and digital return paths are maintained by placing the circuitry on different parts of the card.

On the ADS card, analog components vulnerable to noise is only present inside mixed mode ICs, which makes a normal ground plane a proper design.

Decoupling Capacitors

Active components, and in particular digital components, are drawing power unevenly and in pulses. This leads to big transient currents, voltage drops and noise. Decoupling capacitors, also named bypass capacitors, are placed as local buffers for the power supply or to suppress high frequency noise. A typical high frequency capacitor has a capacitance of $100nF$, and a low frequency capacitor to maintain the supply voltage is normally $1-100uF$. Bypass capacitors must be positioned with care, to achieve the best performance. The capacitor must be located close to the ICs power pins, the transient currents will now flow the short distance between the ICs power and ground pin, through the capacitor. Since a clean ground is important, it should be prioritized to place the capacitor close to the ground pin rather than the power pin. Ideally, the capacitor is placed closer to the IC pins than the connection to the power and ground planes[32].

On the ADS card, $100nF$ high frequency ceramic capacitors are placed close to IC pins, while $100uF$ low frequency tantal capacitors are working as a buffer for groups of components, logically and physically close to each other.

Power Plane

On the ADS card, the power layer is separated into three different power planes. One regular power plane covering most of the card ($3.6V$), and a spitted field over the SAR gyros area. The splitted field consists of a $3.6V$ area and a $5V$ area. The $5V$ area is connected to the output of the voltage converter, and is the power for the SAR sensors and the $5V$ part of the voltage-level translators. The $3.6V$ area is connected to the main power plane through a jumper, and is supplying the voltage converter and the $3.6V$ part of the voltage-level translators. The reason for separating the power from the rest of the card by a jumper is the ability to disconnect or measure the power consumption of the SAR sensor part. This design is adversely due to the extra length of the signal return path, and should not be present in a final version. The jumper was intentionally implemented since the card was designed among other to evaluate the SAR sensors, and its power consumption.

4.3 Mini Backplane Card

The ADCS card is designed to be connected to the Back Plane Card in the satellite. Hence, the card is not suited for operation alone. The ADCS card does not have power connector/regulator nor a PC interface connector, such as RS-232 or Universal Serial Bus (USB). To achieve these properties, a Mini Backplane Card was developed. The Mini Backplane Card is a small $67mm * 25mm$ four layer PCB-card, designed to fulfill the lacking futures of the ADCS card. The card is designed to be reused on any module card designed for the satellite. The Mini Backplane Card includes the following components:

- Power management including.

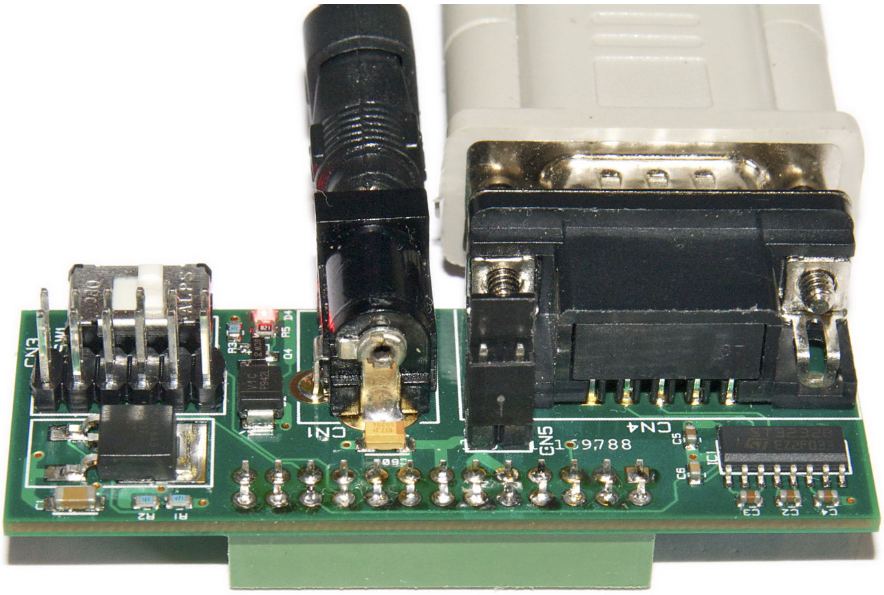


Figure 4.11: The Mini Backplane Card. The connection of a CubeSTAR Module Card to a power source and a PC is made easy and reliable

  - Standard DC socket, 2.1/5.5mm.
  - Power regulation.
  - Power switch.
  - Jumper to connect a multimeter for power consumption measurements.

- RS-232 interfacing including.
  - 9-pin D-sub female connector.
  - UART to RS-232 level converter.

- Pins to connect to the unused communication lines on the backplane connector.

The power regulator and connector makes a regulated power supply needless, and a cheap small "power adapter" is sufficient as a power source. The Mini Backplane Card also protects the ADCS card against destructive voltages to the card caused by incorrect use or connection.

4.4 Microcontroller Firmware

4.4.1 Firmware Development for the AVR Platform

The firmware developed for the microcontroller is written in C code language. The AVR Libc C library is utilized, which is a library developed for the GNU Compiler Collection (GCC) compiler and AVR microcontrollers. The WinAVR package contains the library, compiler (avr-gcc) and definition files for Atmel microcontrollers including the ATXMEGA128A1. For several of the microcontroller peripherals, like UART, SPI and TWI, the device drivers provided in Atmels Application Notes for UART[33], SPI[34]and TWI[35] are utilized. The Atmel AVR Studio 4 is used as programming and debugging platform together with the Atmel AVR JTAGICE mkII. The mkII is a USB connected device for connecting the PC to the microcontroller via JTAG or Program and Debug Interface (PDI). In this thesis PDI is utilized in favor JTAG.

4.4.2 Program Flow and State Machine

The main part of the microcontroller firmware is based upon a Finite State Machine (FSM), which provides a clear overview of the program flow. In total eight states are defined, each state having a corresponding function in the code. The logic deciding the next state is located in the end of each function. A state diagram 4.12 illustrates the eight states and its possible transitions. As seen in the diagram, the microcontroller can be in one of four different modes, *idle, sample*, *bDot* or *External Control.* The mode is determined by external control, either by UART, or in future revision from the satellites OBDH. A sample timer is initialized and running on the microcontroller. Except a manual single sample, the timer has to make an interrupted before the *Sample* state can be entered. The *Idle* and *External Control* state functionality are not implemented in this thesis. The *Idle* state can be used to put the microcontroller and external sensors in sleep to save power, and the *External Control* state can send sensor data and receive magnetorquer control data. Since the rest of the ADCS and OBDH are not yet developed, the functionality of this state is a subject of change.

Additional to the functions which represents each state, several interrupt handler functions and initialization functions are present. In 4.1, the initialization function calls in *main()* is listed. The listing shows the initialization process when the microcontroller starts or have been reset. Each of the *init_* functions is specific to the hardware internally in the microcontroller or to the external components. After initialization, a short while-loop listed in 4.2 executes the corresponding function for the current state, this is the core of the state machine. To fully understand the function call, the *main.h* should be examined, which shows an array of all corresponding states and functions.

Listing 4.1: Initialization function call in main() function.

```
init_clk();      // System clock
_delay_ms(50);
init_RTC32();    // Real time clock
init_uart();     // UART ports
init_hmc5883();  // Magnetometer HMC5883
init_itg3200();  // Gyro sensor ITG-3200
init_sar150();   // Gyro sensors SAR150
init_adc();      // Analog-Digital Converter
init_coils();    // PWM output for coil control
//Sample timer, def.: 500 ms*3.6=1800
init_SampleTimer(SAMPLE_TIMER, 1800);
```

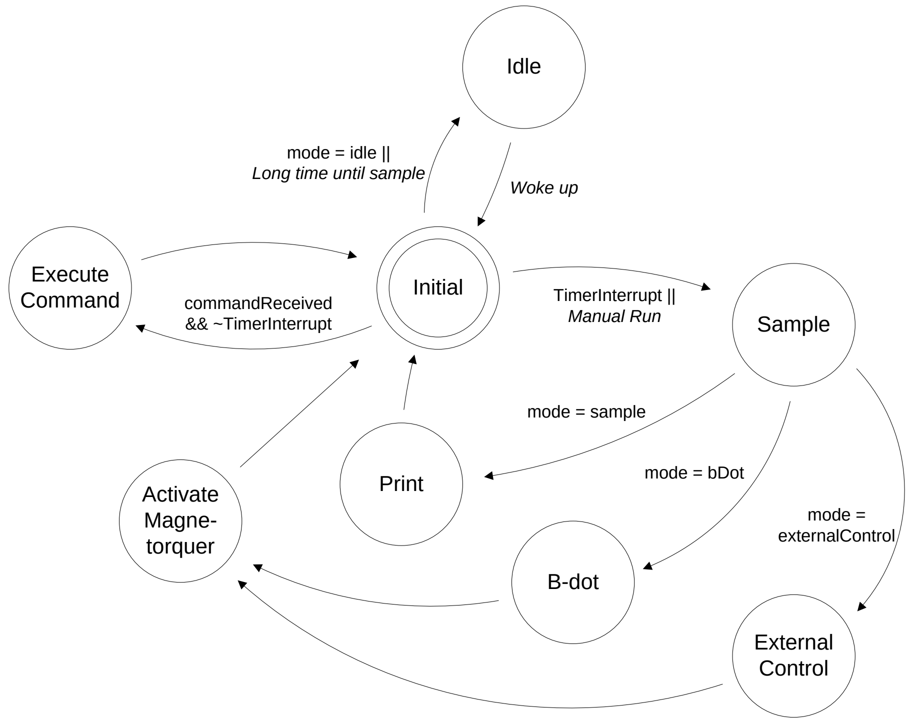


Figure 4.12: State diagram of the microcontrollers Finite State Machine.

Listing 4.2: While(1) loop in main function. The loop is the core of the State Machine, calling the right state function.

```c
while(1) {
    //As long as a function is defined, loop
    for(uint8_t i=0; menu_state[i].pFunc ;i++) {
      //Find state array number of right state
      if (state == menu_state[i].state) {
        //Run function with corresponding array number.
        state = (menu_state[i].pFunc)();
        break;
      }
    }
  } //while end
```

4.4.3 Sensor Drivers

For each of the three sensor models utilized in this thesis, a hardware driver library is developed from scratch. Each of the libraries consists of a code file (.c) and header file (.h). The header files defines all registers and parameters of the chips, additional to creating proper data types for further object oriented programming. The sensors data bus *SPI* or I^2C are implemented utilizing the device drivers from Atmel. The creation of a driver library for each sensor enables reuse, and ensures a firmware which is easy to modify in the future. In all of the sensors, two separate 8-bit registers internally on the sensors contained a 16-bit signed measurement value. A data type reflecting this was created by combining a *union* and a *struct* as showed in 4.3. Each of the sensor drivers contains functions which reads and writes registers to the sensor, which is the essential purpose of these drivers.

Listing 4.3: Data structure from the SAR150 h-file. By utilizing union, the int16_t is allocated to the same memory location as the two uint8_t msb and lsb.

```
typedef union sar_rate
  {
    struct
      {
          uint8_t lsb;
          uint8_t msb;
      } b2;
    int16_t i16;
  } sar_rate_t;
```

4.4.4 UART / RS-232 Control

Control and monitoring of the microcontroller is performed via the UART/RS-232 interface. The microcontroller is sending and receiving characters on the UART port, which is level converted on the Mini Backplane Card. A 9-pin D-sub cable is connected to the PC, and control is available through terminal programs like Microsoft HyperTerminal, or the custom control software explained in 4.5. The communication protocol is implemented on the microcontroller using the interrupt based USART device driver from Atmels Application Note 1307 combined with the libc implementation of *stdio* (Standard input/output), a part of the C standard library. The implementation of the stdio library, enables an easy and powerful print functionality in the code.

The communication is performed by sending ASCII words, optionally followed by a value also expressed in ASCII. Echo is implemented on the microcontroller, so that HyperTerminal easily can be utilized, without having to turn on local echo, which is not a default setting. A control command sent to the microcontroller must be in the following ASCII format: *<command> <optional value><CR>*. The Carriage Return (CR) is sent from HyperTerminal by pressing the *return* key. Return messages from the microcontroller are sent in the following format: *<message><CR><LF>*, which makes HyperTerminal start from the beginning of a new line after the message is written. In 4.3, all commands are listed. The *coil*<axis> command is adjusting the direction and percentage *on*, out of the h-bridge circuit. According to the formula in the referred table, *coilx 0* will set the the coil to -100%, or in words, fully on, in reverse direction.

Command	Description
`reset`	Full reset of the microcontroller.
`magstart`	Sets sample flag on HMC5883 magnetometer
`magstop`	Clears sample flag on HMC5883 magnetometer
`sarstart`	Sets sample flag on SAR150 gyro
`sarstop`	Clears sample flag on SAR150 gyro
`itgstart`	Sets sample flag on ITG-3200 gyro
`itgstop`	Clears sample flag on ITG-3200 gyro
`idle`	Set microcontroller mode to *Idle*
`sample`	Set microcontroller mode to *Sample*
`bdot`	Set microcontroller mode to *B-Dot*
`extcont`	Set microcontroller mode to *External Control*
`rate <value>`	Time between samples Range: $0 - 1000$ Unit: ms
`coilx <value>`	Percentage PWM signal, x-axis coil Range: $0 - 254$ Unit: $\frac{(x-127)}{127} * 100\%$
`coily <value>`	Percentage PWM signal, y-axis coil Range: $0 - 254$ Unit: $\frac{(y-127)}{127} * 100\%$
`coilz <value>`	Percentage PWM signal, z-axis coil Range: $0 - 254$ Unit: $\frac{(z-127)}{127} * 100\%$
`status`	Returns mode, sample rate and sensor sample flags

Table 4.3: Command set for the ADCS microcontroller. All commands are case sensitive.

4.4.5 Response Messages

Some actions in the microcontroller is sending a message to the UART port. The messages sent include status, measured data and response to some of the control commands. Some of the messages are easy to interpret by humans, while some are designed to be read by a software on the PC. All strings sent by the microcontroller are followed by a *Carriage return* (0x0D) and *Line feed* (0x0A) ASCII characters, which performs a line shift on HyperTerminal. In 4.4 all response messages which can be sent by the microcontroller are listed. The variables in the messages are denoted by the C-languages printf syntax scheme. A variable starts with a "%" and ends with a letter describing representation method, in our case x for hexadecimal representation and d for decimal representation. The number in between, sets how many characters to be utilized, and if zero padding should be enabled. The common representation mode *%04x* describes a hexadecimal four character number, such as *00A4*.

Name	Description	Message	
		Scale factor / variable information	Unit
SAR data	Printf syntax	SAR:%04x%04x%04x%04x%04x%04x%04x	
	Described syntax	SAR:<X-axis><Y-axis><Z-axis><temp X><min><ms>	
	X-axis data	0.1	deg/s
	Y-axis data	0.1	deg/s
	Z-axis data	0.1	deg/s
	Temperature	1	°C
	RTC minutes	1	minutes
	RTC ms	1	milliseconds
ITG data	Printf syntax	ITG:%04x%04x%04x%04x%04x%04x%04x	
	Described syntax	SAR:<X-axis><Y-axis><Z-axis><temp X><min><ms>	
	X-axis data	1/14.375	deg/s
	Y-axis data	1/14.375	deg/s
	Z-axis data	1/14.375	deg/s
	Temperature	1/280	°C
	RTC minutes	1	minutes
	RTC ms	1	milliseconds
HMC data	Printf syntax	SAR:%04x%04x%04x0000%04x%04x%04X	
	Described syntax	SAR:<X-axis><Y-axis><Z-axis><temp X><min><ms>	
	X-axis data	1/13000000	Tesla
	Y-axis data	1/13000000	Tesla
	Z-axis data	1/13000000	Tesla
	RTC minutes	1	minutes
	RTC ms	1	milliseconds
Status	Printf syntax	STA:%04d-%1d-%1d-%1d-%1d	
	Described syntax	STA:<Smpl rate>-<mode>-<hmcSmpl>-<itgSmpl>-<sarSmpl>	
	Sample rate	1/3.6	milliseconds
	Mode [0-4]	0: idle 1: sample 2: bDot 3: externalControl	Enum
	hmcSample [0-1]	1: true 0: false	Bool
	itgSample [0-1]	1: true 0: false	Bool
	sarSample [0-1]	1: true 0: false	Bool
Rate set	Printf syntax	Time between sample is: %d ms	
	Described syntax	Time between sample is: <rate> ms	
	Rate	1	milliseconds
Startup	Message	Welcome!	
Rate error	Message	Rate must be set between 0 and 1000	
Syntax error	Message	Syntax error	

All values described %04x are ascii/hex represented 16-bits, two's complement values
All values described %___d are ascii represented decimal values

Table 4.4: Description of response messages from the ADCS card.

4.5 LabView Interface VI

To be able to perform calibration procedures, and for measurement data validation, it was important to create a PC side application as a companion to the ADCS card. A Graphical User Interface (GUI) was desired, since it have the benefit of being easier to learn and faster to use for new users. A program, or Virtual Instrument (VI) which is the equivalent name in the LabView domain, was developed. The VI communicates through a serial interface to the microcontroller utilizing the same command set and response messages as described in 4.3 and 4.4. LabView is a visual programming language, well suited for data acquisition and representation. Developing the logic of a program is performed by drawing a Block Diagram, and a GUI by drawing the desired elements in a LabView Front Panel.

The VI developed, named "ADCS interface.vi" is found in appendix C, but should preferably be examined and tested digitally. The most important features of the VI developed are the following:

- Control and status polling of mode, sample rate and sensor sample flag of the microcontroller.
- Graphical representation of measurement values from all of sensors
- Ability to store measured value
- Sensor calibration functions
- 3D-representation of magnetometer data
- Control of the rate table for gyro calibration process

The VI is utilizing two serial ports. For both to work, the right PC serial port on the computer must be chosen in the GUI of the VI.

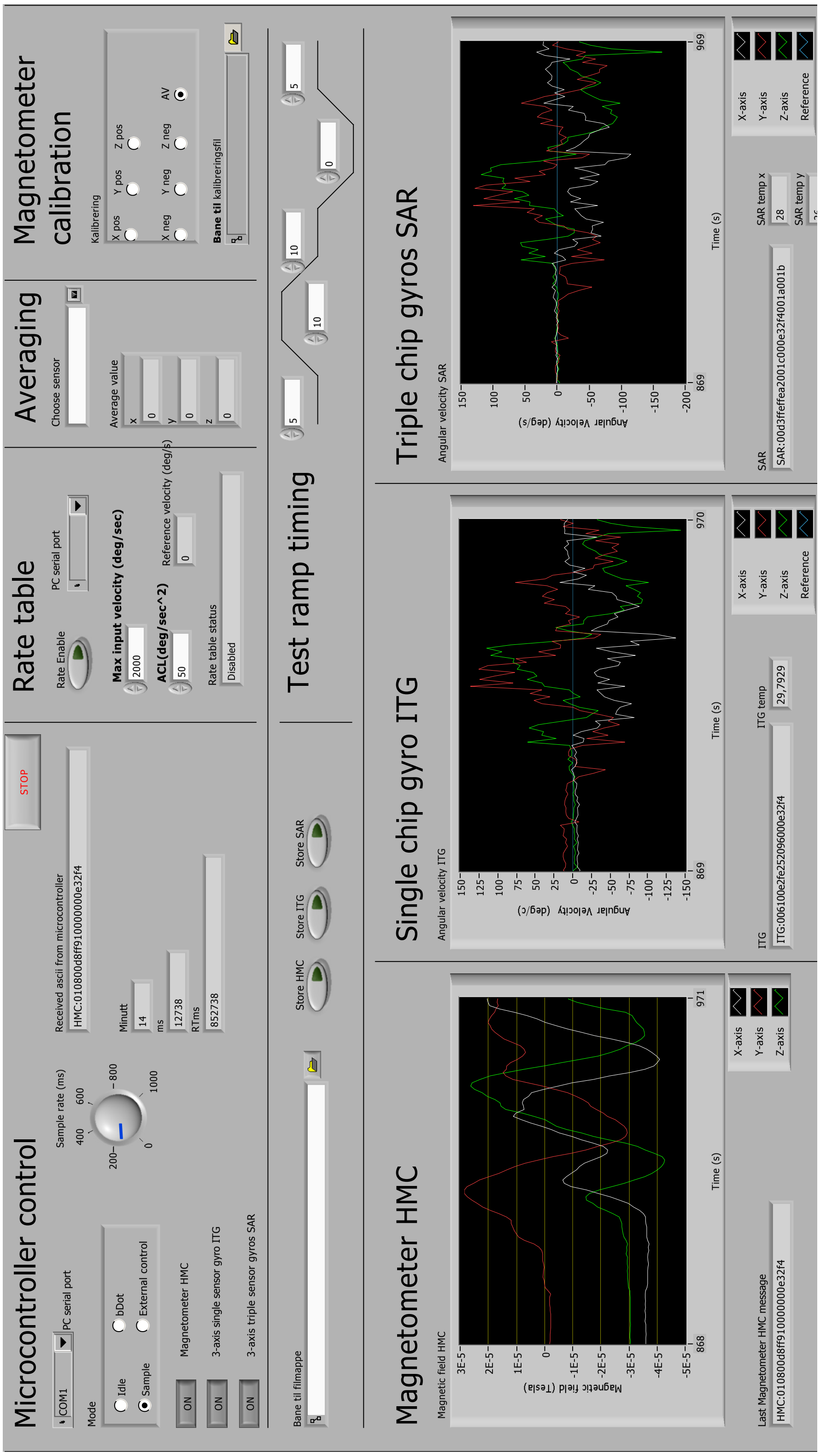


Figure 4.13: Screenshot of the Front Panel of the LabView VI developed to test the ADS card

Chapter 5

Sensor Calibrating

We want to determine how good our sensors performs, and try to correct error sources as good as possible. Even if the errors can not be corrected, it is desired to know the performance of each sensor. An error model of a sensor gives us knowledge to determine if the sensor is suitable for its purpose, and can be very useful in simulation of the system. Many error sources can easily be corrected for, when known. In this chapter, we are characterizing the error of the sensors utilized in the thesis, and establish practical methods for calibration of the sensors.

5.1 Gyro

Two different gyro sensor setups are tested in this thesis, a single chip three-axis ITG-3200 gyro sensor, and the SAR150 setup consisting three one-axis high precision gyros. It is interesting to see how the sensor setups perform compared to each other. The orthogonal setup of the SAR150 ICs are mounted by hand, and it is also interesting to measure the accuracy of this mounting. The gyro calibration methods in this thesis are based on the methods presented in [36] and [37] by Bekkeng, J. K.

5.1.1 Error Characterization

The most significant errors of a gyro sensor are:

- Scale factor (λ)
- Misalignment (δ)
- Random bias (η)
- Temperature dependent bias ($O(T)$)

Where the denoting letters are listed in brackets. We are defining a misalignment and scale factor *error* matrix M, where δ_{ij} is representing the projection of the sensitive axis i on the body axis j, given in radians. The sensitive axis x, y and z of the gyro are intended to be in the same directions as the corresponding body frame axes, hence the misalignment angle is assumed small. M is defined as

$$M = \begin{bmatrix} \lambda_x & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & \lambda_y & \delta_{yz} \\ \delta_{Zx} & \delta_{zy} & \lambda_z \end{bmatrix}$$

We also defines a misalignment and scale factor matrix S as

$$S = I + M = \begin{bmatrix} 1+\lambda_x & \delta_{xy} & \delta_{xz} \\ \delta_{yx} & 1+\lambda_y & \delta_{yz} \\ \delta_{zx} & \delta_{zy} & 1+\lambda_z \end{bmatrix}$$

We assume a linear temperature dependency of the bias, which gives

$$\mathbf{O}(T) = \mathbf{O}_0 + \mathbf{a}_T T \tag{5.1}$$

which makesa_T the temperature coefficient. $\mathbf{O}_0$ is the offset at $0°C$.

As a result of the above definitions, we have that the true angular rate ω^b in the body frame has the following relations to the measured angular rate ω^s

$$\omega^b = S(\omega^s - \mathbf{O}(T)) - \eta_v$$

where the v in η_v denotes that the noise is zero-mean Gaussian white noise.

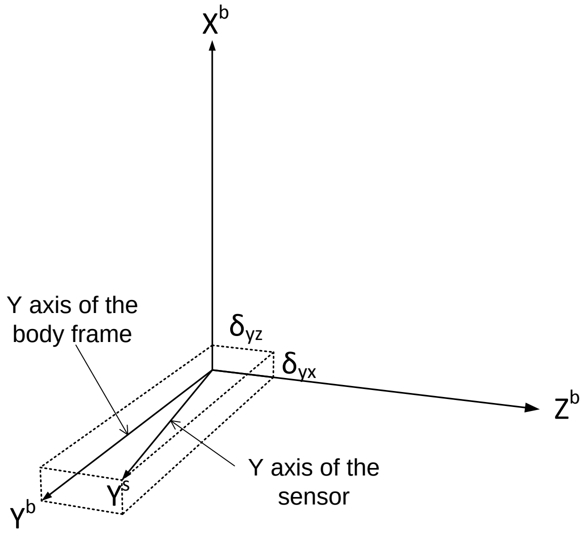


Figure 5.1: Illustrating of the definition of the small misalignmentδ.

5.1.2 Temperature Bias Calibration

We are later going to determine M in a calibration process explained in section 5.1.3, but first we want to find the temperature dependent bias$\mathbf{O}(T)$. The bias is measured by measuring the sensors when not in motion. To get the temperature dependency, we measured the bias in a wide temperature range, wile logging the temperature from the internal temperature sensors of the chips. Two experiments were performed:

5.1.2.1 Freezer Test

The ADS card was placed inside a regular freezer with a temperature of approximately $-20°C$. The card was connected to computer outside of the freezer, running the LabView VI as described in section 4.5. The sampling was started immediately after placed inside the freezer, at a sampling rate of $0.5Hz$. When the temperature sensors inside the gyros was close to the freezer temperature, the sensors was taken out and sampling was continued as the chip altered room temperature.

5.1.2.2 Oven Test

In the same way as the freezer test, the ADS card was placed inside a regulated oven. The temperature was raised to $80°C$, wile sampling at the same frequency, $0.5Hz$. When the sensors reached the temperature of the oven they where placed in room temperature, while the measurement still was ongoing. Because of a fan inside the oven, making disturbing vibrations, only the sampled data of the cooling from $80°C$ down to room temperature was usable.

5.1.3 Reference Data Acquisition

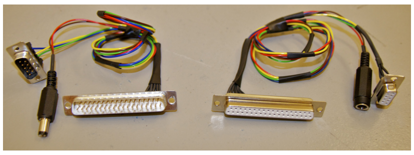


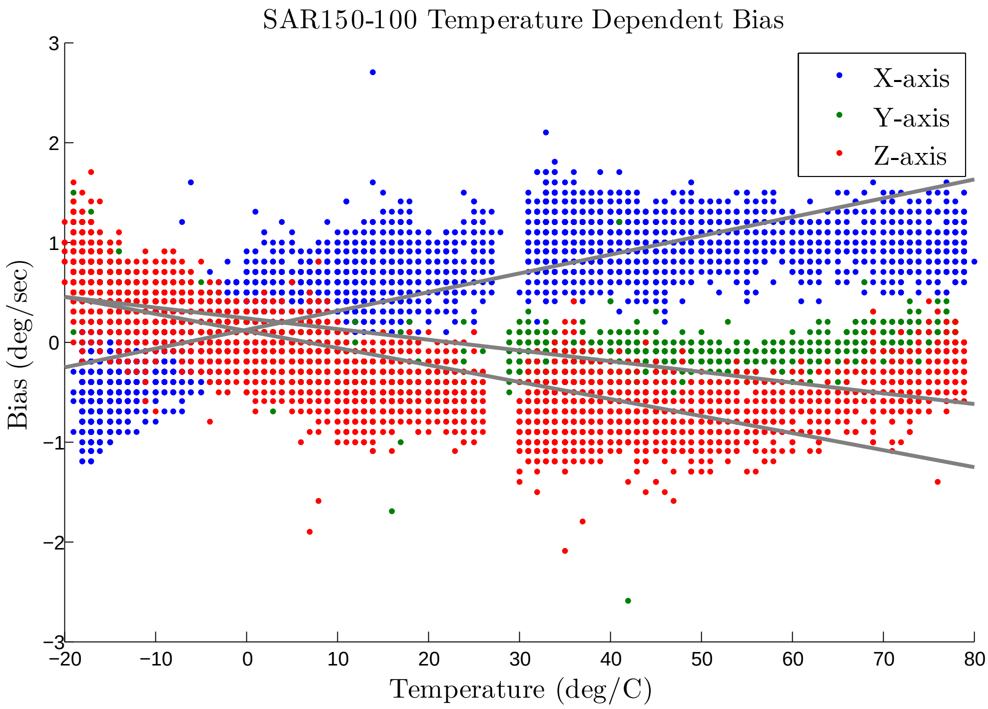
Figure 5.3: Adapter cable for the ADCS card and Rate Table

To be able to determine M, we need appropriate reference data. A controlled rotation in one axis at a time is desired, and an experiment on a reference rate table was performed. The reference table utilized was an Ideal Aerosmith 1291BR Single-Axis Positioning and Rate Table, ideal for this test. The reference table is controlled by an accompanying control unit, which again is controlled through a RS-232 interface. The control unit returns the actual angular velocity of the rate table throughout the process. The ADCS LabView VI was programmed to also control the rate table. The VI makes a coherent interface for the whole experiment, and makes sure reference data from the rate table are acquired at the same time as from the sensors.

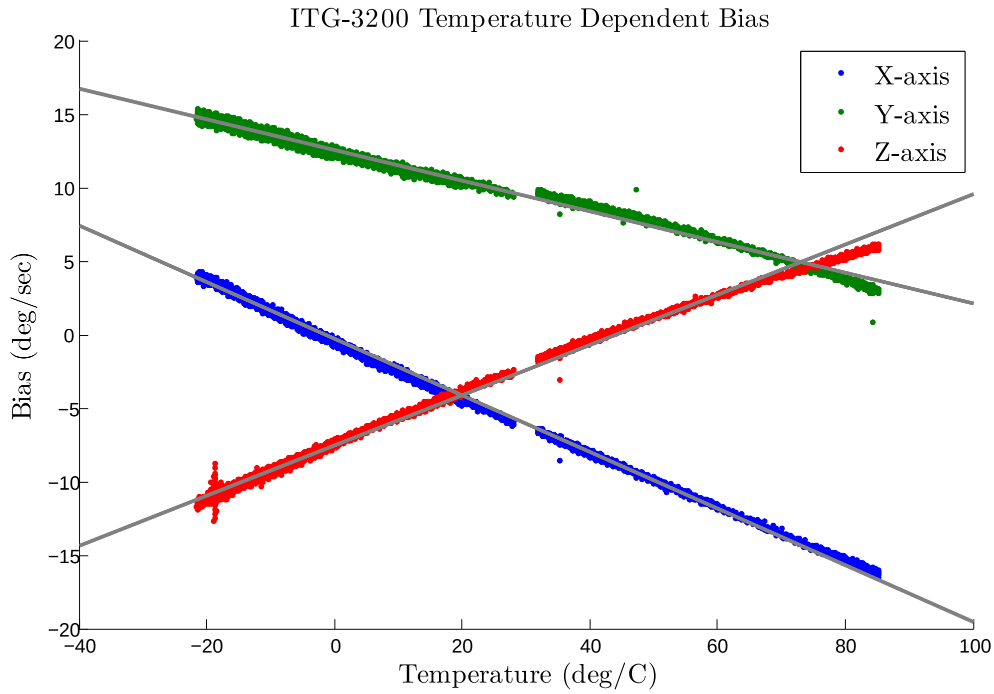
Physically, a method for mounting the card in all three orthogonal directions had to be developed. Since the satellite structure already existed, and is constructed to hold the card, a mounting bracket to attach the satellite structure to the spin table was manufactured at the mechanical workshop. The bracket is constructed so the structure can be mounted horizontal and vertical, in that way, we can spin the card around all three axes. The spin table has two 37 pin D-sub connectors located on the rotating table, which is connected to corresponding connectors on the fixed base. Two adapter cables were made to send power and RS-232 signal through the D-sub connectors.

5.1.4 Kalman Filtering

In this section, a Kalman filter is utilized. A good introduction is found in [39], while a comprehensive review is given in [38]. A simple Kalman filter was utilized to find the optimal parameters based on the reference test. To use the Kalman filter, we first have to model the parameters we want to estimate in a way that suits the Kalman filter. We



(a) The SAR150-100 sensors have a low bias, with a low temperature dependency. The graph consists only of dots, which is caused by the resolution of the data from the sensor. By visually examine the shape of the plot, a second order fit would probably made an improvement, but is not performed in this thesis.



(b) The ITG-3200 have a high bias, but it is highly linear. Note that the z-axis temperature coefficient is opposite signed than the two others. This is probably an effect of the slightly different design, as described in section 2.2.2.2.

Figure 5.2: The plots shows the bias on the three sensor axes at temperatures ranging from $-20°C$ to $80°C$. There is a visible gap between the freezer test (left) and the oven test (right). No measurements are done in this small temperature range.

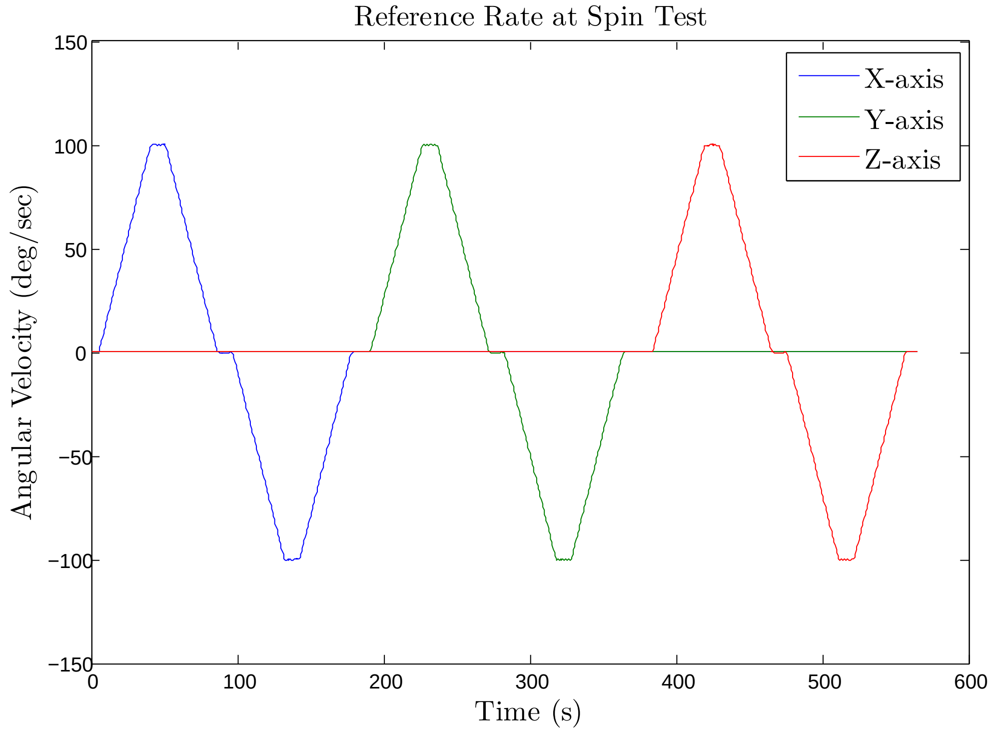


Figure 5.4: Reference Rate in the Spin Test. Three separate single axis test cycles are merged together after each other as one continuous test. This is how we work with the data after the test.

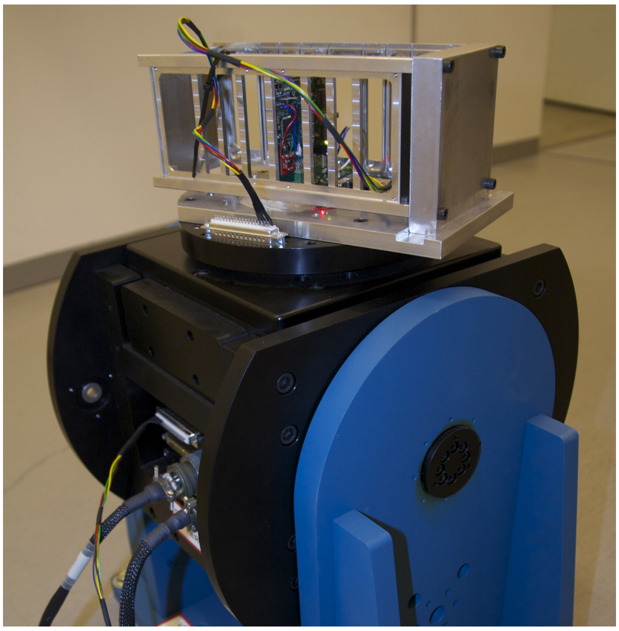


Figure 5.5: The ADCS card mounted and connected on the rate table. The mounting frame and the satellite structure enables the card to be mounted right. The many-colored cable are made custom for the ADCS card and rate table.

assume that we already have utilized equation 5.1 in a *pre-processing*, so the data is temperature compensated.

We defines $\tilde{\omega}$ as the per-processed measurement data in the sensor frame. The true angular body frame rate is then given by

$$\begin{aligned}\omega^b &= \mathbf{S}\tilde{\omega} - \beta - \eta_v \\ &= (\mathbf{I} + \mathbf{M})\tilde{\omega} - \beta - \eta_v \\ &= \tilde{\omega} + \Omega m - \beta - \eta_v \end{aligned} \tag{5.2}$$

where

$$\mathbf{\Omega m} = \mathbf{M}\tilde{\omega}$$

with

$$\mathbf{\Omega} = \begin{bmatrix} \tilde{\omega}_y & \tilde{\omega}_z & 0 & 0 & 0 & 0 & \tilde{\omega}_x & 0 & 0 \\ 0 & 0 & \tilde{\omega}_x & \tilde{\omega}_z & 0 & 0 & 0 & \tilde{\omega}_x & 0 \\ 0 & 0 & 0 & 0 & \tilde{\omega}_x & \tilde{\omega}_y & 0 & 0 & \tilde{\omega}_z \end{bmatrix}$$

and

$$\mathbf{m} = \begin{bmatrix} \delta_{xy} & \delta_{xz} & \delta_{yx} & \delta_{yz} & \delta_{zx} & \delta_{zy} & \lambda_x & \lambda_y & \lambda_z \end{bmatrix}^T$$

The state vector of the Kalman filter is defined as

$$\mathbf{x} = \begin{bmatrix} \mathbf{m} \\ \beta^b \end{bmatrix}$$

so the *Kalman state function* can be modeled as

$$\mathbf{x}_{k+1} = \mathbf{x}_k$$

where the k denotes the state. The fact that the values of the state actually not change, makes the Kalman filter fairly simple. The measurement $\mathbf{z}$ is defined as the difference from ω^b, given by the rate table, to the measured value:

$$\mathbf{z} = \omega^b - \tilde{\omega} = \mathbf{\Delta}\omega$$

Since $\mathbf{z}$ is the *difference,* we can remove the $\tilde{\omega}$from 5.2, and define the Kalman measurement equation as

$$\mathbf{z}_k = H_k \mathbf{x}_k + \mathbf{v}_k$$

where H is a 3×12 element matrix given by $H = \begin{bmatrix} \mathbf{\Omega} & -I_{3\times3} \end{bmatrix}$, and $\mathbf{v}$ is the zero-mean Gaussian noise vector.

5.1.5 Matlab implementation

The pre-process filter and the Kalman filter were implemented in Matlab, as GUI software. The program utilizes the data from the static temperature test and the spin table test. The whole process is performed only by choosing the right data sources. The Matlab program is designed to fulfill the calibration chain developed, and have been designed to read the automatically generated LabView data files directly. The only "manual" process for the user is to create a single variable of both the freezer test and the oven test. This is performed manually, since experiences showed that there can be some parts of the data which should be rejected.

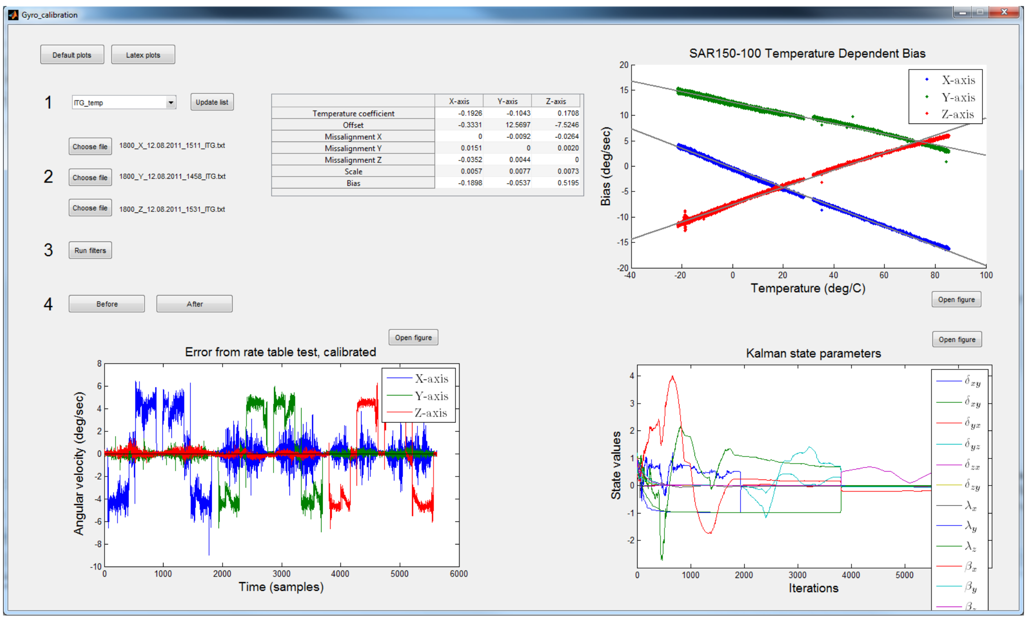


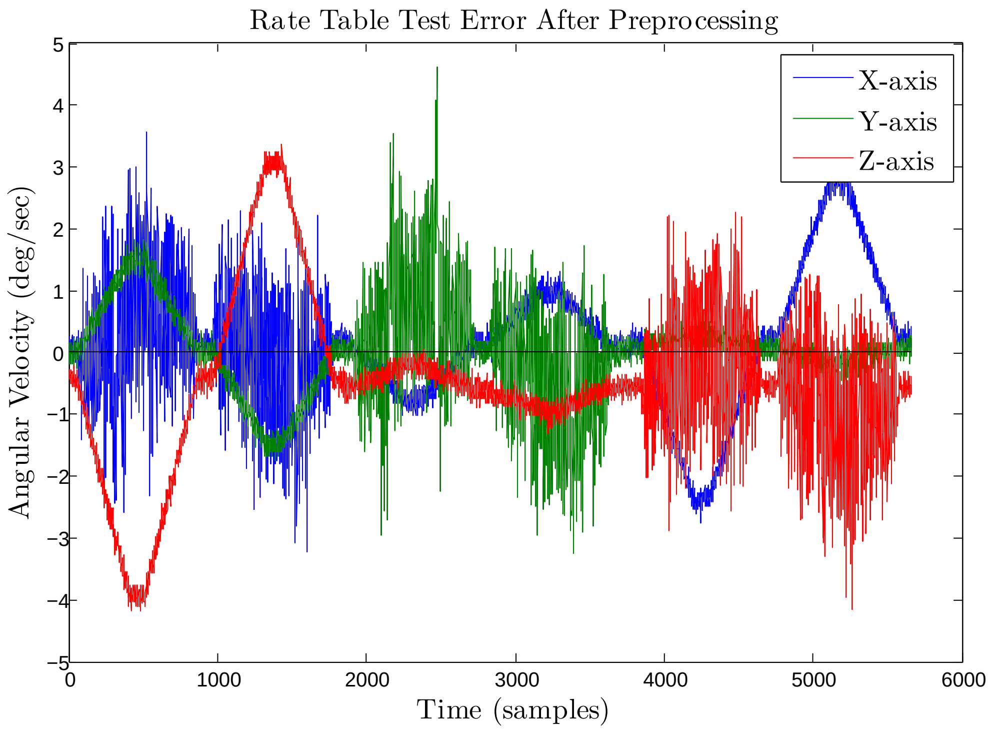
Figure 5.6: A Matlab software with a GUI was developed, to easily perform gyro calibration. All of the calibration values are listed in a table.

5.1.6 Results

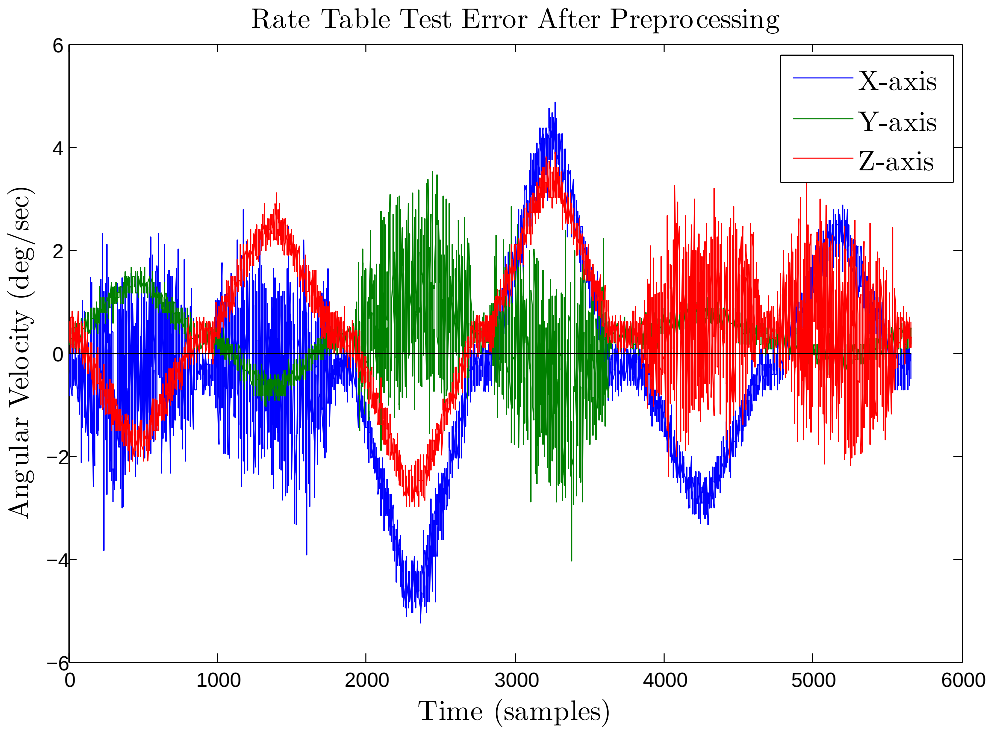
Our version of the SAR sensor, can measure a maximum angular rate of 100°/s, while the ITG-3200 are able to measure up to 2000°/s. The spin table test was performed at both 100°/s and 1800°/s as maximum speed. A selection of the results are presented as figures and tables in this section. Initially, the ITG-3200 sensor provided poor results, compared to the SAR sensors. A high bias was present, and as we observed at the temperature test, it was also highly temperature dependent. However, after a calibration process, the two different sensor setups performed very close to each other, almost unable to distinguish from each other. It was not tested to use a second order temperature fit on the SAR sensor, but the result would probably not change much. In table 5.1 the standard deviation of the resulting error plot is listed for the different axis. It is not tested how the sensors behave over time, but based on the knowledge from the calibration process, the ITG-3200 sensor is preferable in the future work of the project. The recommendation is a based on the price, physical space on the ADCS card, complexity and power consumption of the two sensor setups.

		SAR, 100°/sec			ITG 100°/sec			ITG1800°/sec		
Parameter	Symbol	X	Y	Z	X	Y	Z	X	Y	Z
Temperature coefficient	a_T	0.0187	-0.0107	-0.017	-0.1926	-0.1043	0.1708			
Pre calibration offset	O_0	0.1244	0.2299	0.1064	-0.3331	12.5697	-7.5246			
Misalignment X	δ_{x}_		-0.0423	-0.0255		-0.009	-0.0259		-0.0092	-0.0264
Misalignment Y	δ_{y}_	0.0101		0.0042	0.0151		0.0018	0.0151		0.002
Misalignment Z	δ_{z}_	-0.0211	-0.0288		-0.0353	0.0041		-0.0352	0.0044	
Scale	S	0.0002	0.0083	0.0021	0.006	0.0087	0.0067	0.0057	0.0077	0.0073
Bias	β	0.3153	-0.3222	-0.4121	-0.1689	-0.0395	0.5258	-0.1898	-0.0537	0.5195
Standard deviation		0.5847	0.5297	0.5513	0.5076	0.5244	0.4915	2.2453	2.1648	2.2520

Table 5.1: Gyro sensor parameters calculated from test data. The calibration results of the temperature calibration and spin test. The values are automatic calculated by the Matlab software developed, all values in °/sec.

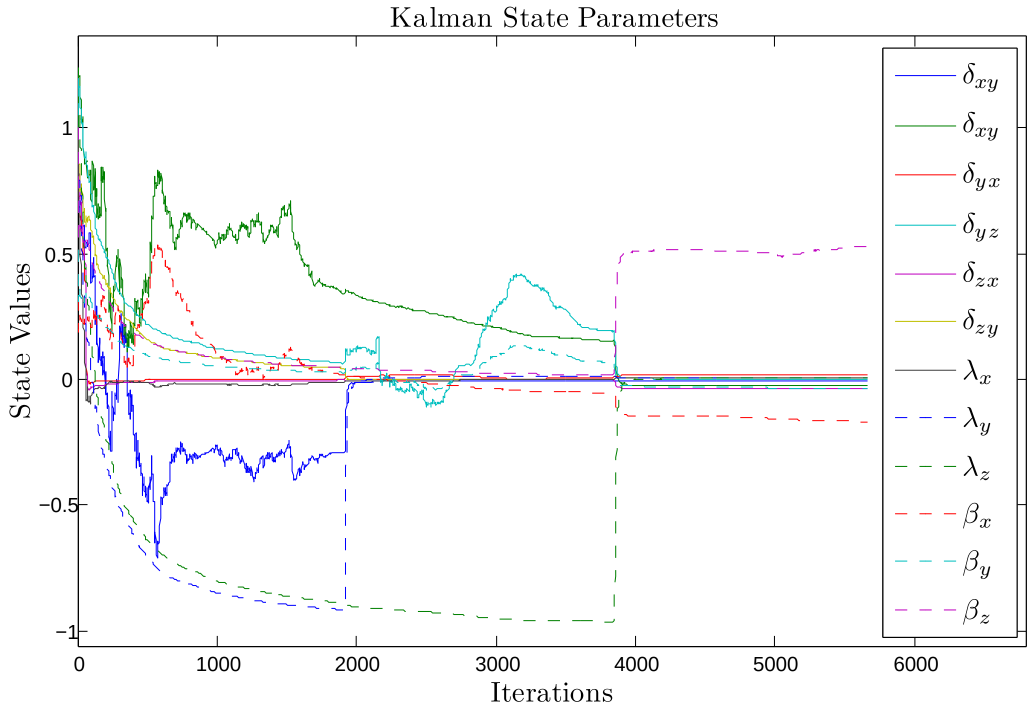


(a) ITG-3200

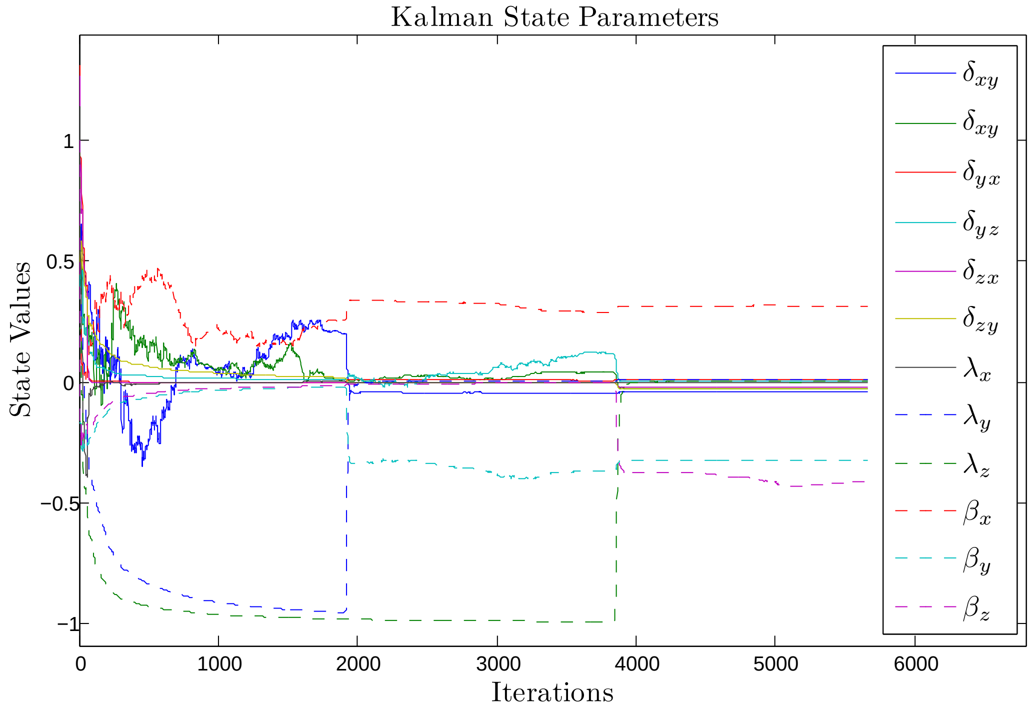


(b) SAR150-100

Figure 5.7: Rate table test error after pre-processing. The measured data from the rate table test is subtracted from the reference and pre-processed. We can clearly see the offset. Remember that the reference is given in figure 5.4. Results of 100°/s test.

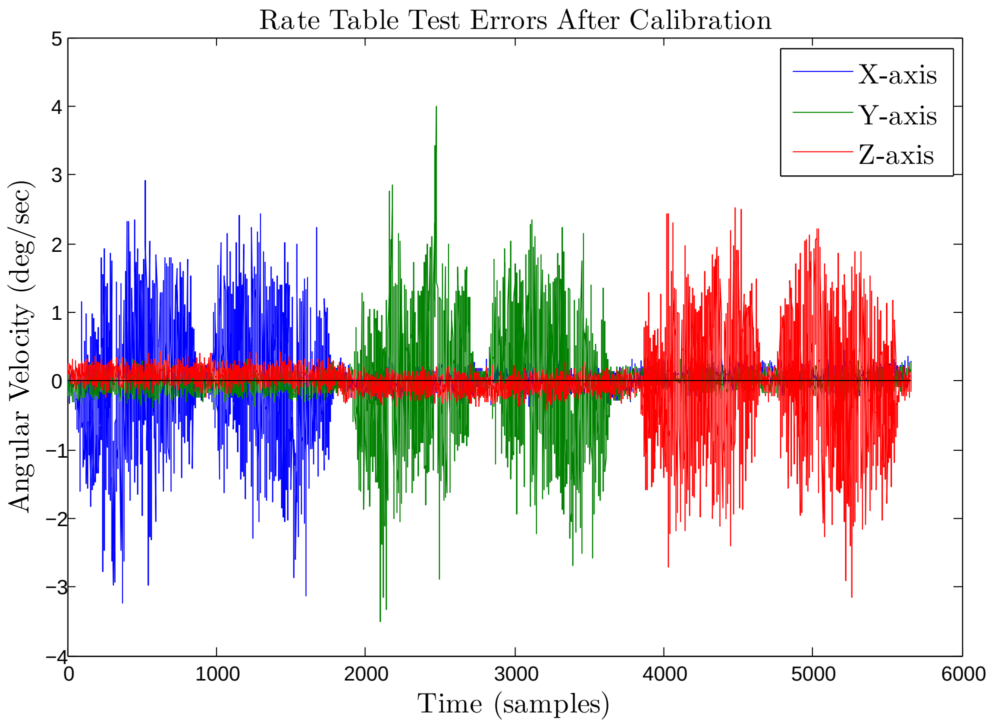


(a) ITG-3200

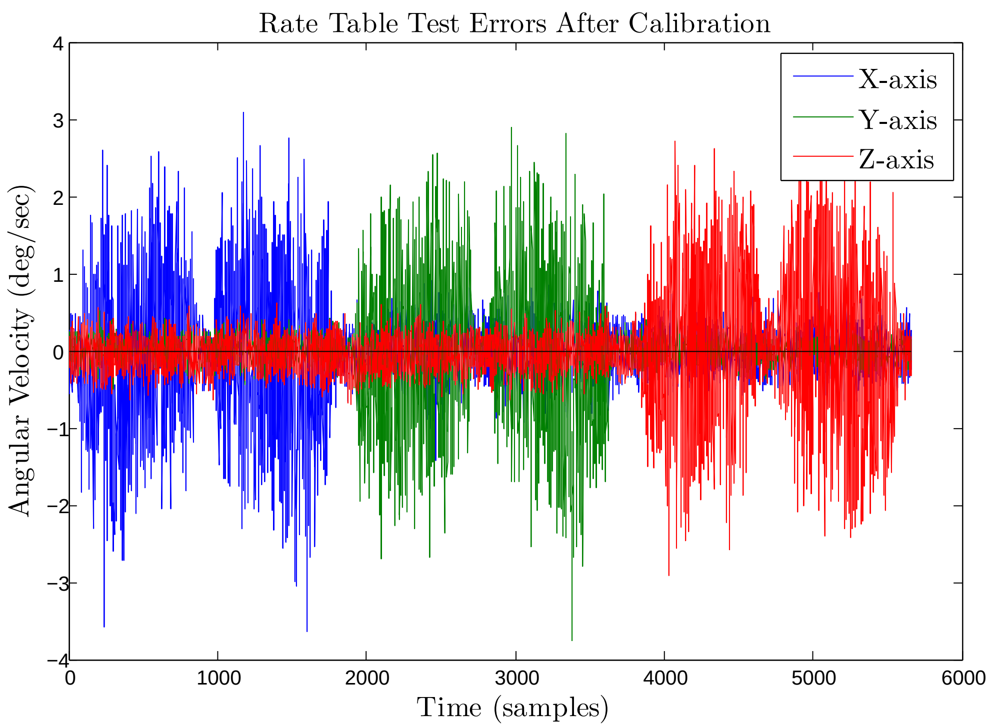


(b) SAR150-100

Figure 5.8: The Kalman filter state parameters during the 6000 iterations. Results of 100°/s test.

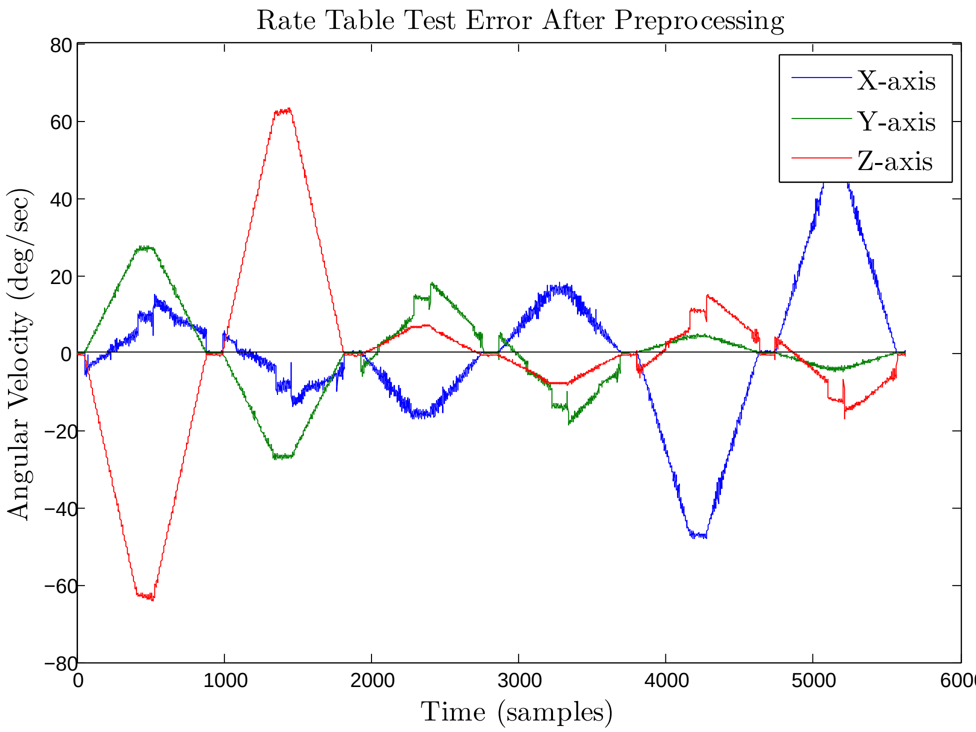


(a) ITG-3200

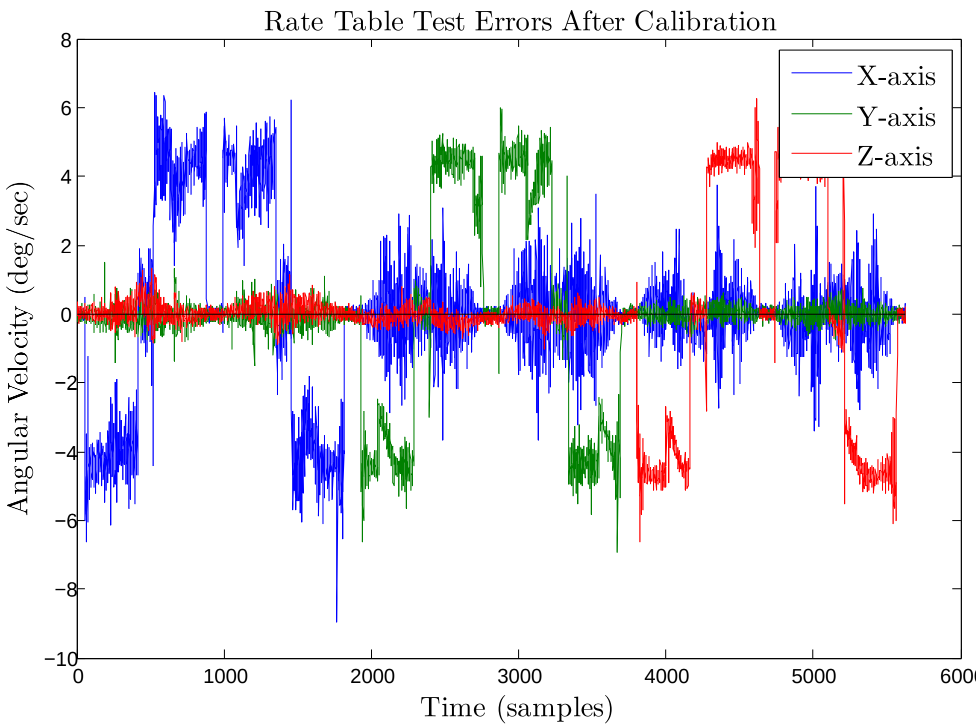


(b) SAR150-100

Figure 5.9: Rate table test error after calibration. The error after pre-processing and calibration is a big improvement from the uncalibrated data set. The performance looks pretty similar on these tables, which is confirmed by the standard deviation values. Results of 100°/s test.



(a) Before calibration



(b) After calibration

Figure 5.10: The ITG-3200 was also tested at a max rate of 1800°/s. The calibration process makes the data way better. An extra bias linear to the acceleration can be observed, and is approximately one tenth of the acceleration.

5.2 Magnetometer

5.2.1 Error Characterization

The magnetometer reading can be distorted by several sources. It is important to know how they affect readings, to avoid and/or compensate for them. The most important error sources for a general magnetometer are characterized and parameterized in this section.

Hard Iron and Soft Iron

Hard iron and soft iron disturbances are changes in the magnetic field applied to the sensor. Hard iron sources are permanent magnets fixed in the frame of the sensor. The disturbing magnetic field from a hard iron source is constant, and creates a magnetic bias. Soft iron sources are ferromagnetic materials, which are being magnetized by the magnetic field applied to it. The magnetic field created is dependent on the direction and magnitude of the applied field. We denotes the hard iron error as $\mathbf{b}_{HI}$, and the more complex soft iron as

$$\mathbf{h}_{SI} = \mathbf{C}_{SI}\mathbf{R}^{be}\mathbf{h}^{e}$$

Where $\mathbf{h}^e$ is the magnetic field in earth (ECEF) frame, $\mathbf{R}^{be}$ is the rotational matrix from body to earth, and $\mathbf{C}_{SI}$ is the 3×3 soft iron transformation matrix.

Scaling and Bias

As on most sensors, a scaling error and a bias must be taken account for. We represents the scale error by the 3×3 matrix $\mathbf{S}_M$, and the bias as by the vector $\mathbf{b}_M \in \mathbb{R}^3$

Noise

A disturbing white uncorrelated noise for each sample is present, and denoted $\mathbf{n}_{mi}$.

Non-orthogonality and alignment error

The non-orthogonality of the three axis of a sensor can also be described by a transformation matrix. We defines

$$\mathbf{C}_{NO} = \begin{bmatrix} 1 & 0 & 0 \\ sin(\psi) & cos(\psi) & 0 \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix}$$

where $(\psi, \theta, \phi)$are the Euler angles Yaw, Pitch and Roll, respectively. The non-orthogonality is seen with the body frame as reference.

All the above errors combined yields

$$\mathbf{h}_{ri} = \mathbf{S}_M\mathbf{C}_{NO}(\mathbf{C}_{SI}\mathbf{R}_i^{be}\mathbf{h}^e + \mathbf{b}_{HI}) + \mathbf{b}_M$$

where $\mathbf{h}_{ri}$ is the non orthogonal magnetic reading. By setting $\mathbf{C} = \mathbf{S}_M\mathbf{C}_{NO}\mathbf{C}_{SI}$, $\mathbf{b} = \mathbf{S}_M\mathbf{C}_{NO}\mathbf{b}_{HI} + \mathbf{b}_M$ and $\mathbf{h}_i^e = \mathbf{R}_i^{be}\mathbf{h}^e$, we have a more general function

$$\mathbf{h}_{ri} = \mathbf{C}\mathbf{h}_i^b + \mathbf{b} + \mathbf{n}_{mi}$$

In [40], it is shown that a Singular Value Decomposition (SVD) performed on $\mathbf{C}$, gives

$$\mathbf{C} = \mathbf{R}_L \mathbf{S}_L \mathbf{V'}_L$$

where the L denotes that they are parameters of an ellipsoid, $\mathbf{R}_L$ is a rotation matrix, $\mathbf{S}_L$ is a 3×3 diagonal scale matrix, and $\mathbf{V}_L$ is a orthogonal transformation matrix. This describes that measurements of arbitrary directed magnetometer in a uniform magnetic field, but with the errors described, will distribute along the surface of a biased, scaled, skewed and rotated ellipsoid. A perfect sensor without error would have distributed its measurement points along the surface of a sphere. Our goal is to find the ellipsoid parameters, to correct back to a sphere. In [40], an optimization problem is defined, to find those parameters. It is shown that by minimizing the unconstrained problem

$$\underset{\mathbf{T}}{min} \sum_{i=1}^{n} (\|\mathbf{T}(\mathbf{h}_{ri} - \mathbf{b}_T\| - 1)^2 \tag{5.3}$$

Where $\mathbf{T}$ is a 3×3 matrix of real values. When the optimal values $\mathbf{T}^*$ and $\mathbf{b}_T^*$ is achieved, a SVD decomposition gives

$$\mathbf{T}^* = \mathbf{V}_L \mathbf{S}_L^{-1} \mathbf{R}'_L$$

which makes that the calibrated value is given by

$$\mathbf{h}_i^c = \mathbf{S_L^{-1}} \mathbf{R}'_L (\mathbf{h}_{ri} - \mathbf{b})$$

There are several elegant methods to solve this minimization problem, and several are mentioned in [40]. For our, not time critical calibration process, and evenly distributed measurements, a numerical brute force method was developed in matlab, and presented in the source code appendix E. The method described by words, like this:

1. Find the best possible values for $\mathbf{T}$ and $\mathbf{b}$, so the process wont take to long. This is performed by looking at the uncalibrated data plot 5.13.
2. Determine the partial derivative for $\mathbf{T}$ and $\mathbf{b}$ in turns, based on the derived formula given in [40]:
$$\nabla \int |_{\mathbf{T}} = \sum_{i=1}^{n} 2c_T \cdot \mathbf{u}_i \otimes \mathbf{T}\mathbf{u}_i$$
$$\nabla \int |_{\mathbf{b}} = \sum_{i=1}^{n} -2c_T \cdot \mathbf{T}'\mathbf{T}\mathbf{u}_i$$
where $\mathbf{u}_i \equiv \mathbf{h}_{ri} - \mathbf{b}$, and $c_T = 1 - \|\mathbf{T}\mathbf{u}_i\|^{-1}$.
3. Correct $\mathbf{T}$ and $\mathbf{b}$ by a small fraction of the corresponding partial derivative.
4. Calculate the error by equation 5.3.
5. Check if the step was successful (less error than last time), and adjust the fraction from 3, up if the error was decreasing and down if not.

If the initial values are not good enough, the process will take very long time, and several hundreds of thousand iterations are necessary. In our case, the optimal parameters was found after approximately 500 iterations, as illustrated in figure 5.11

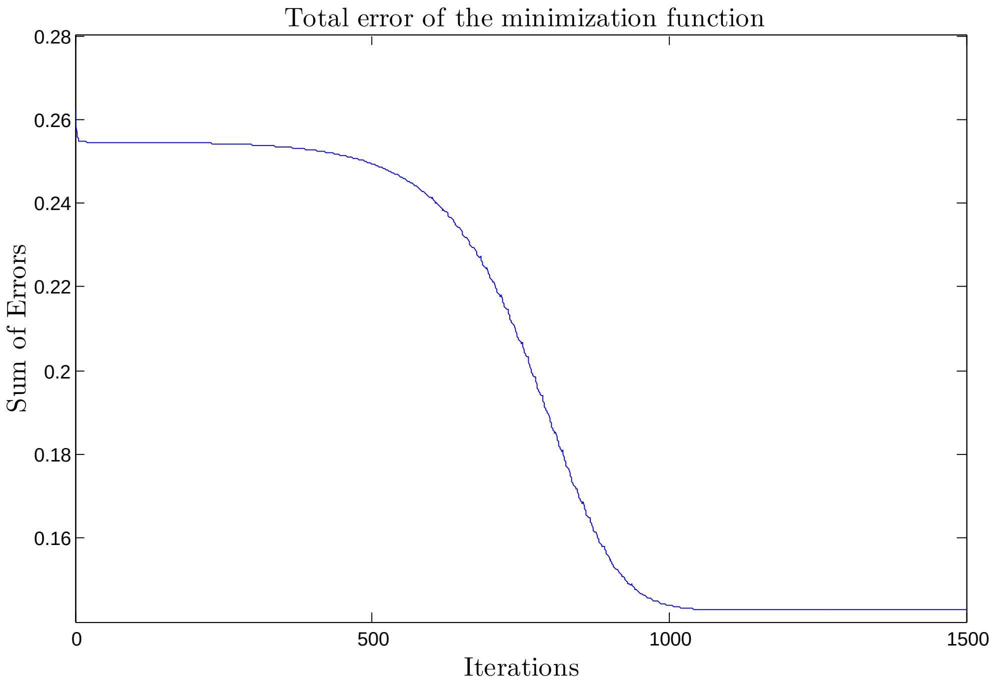


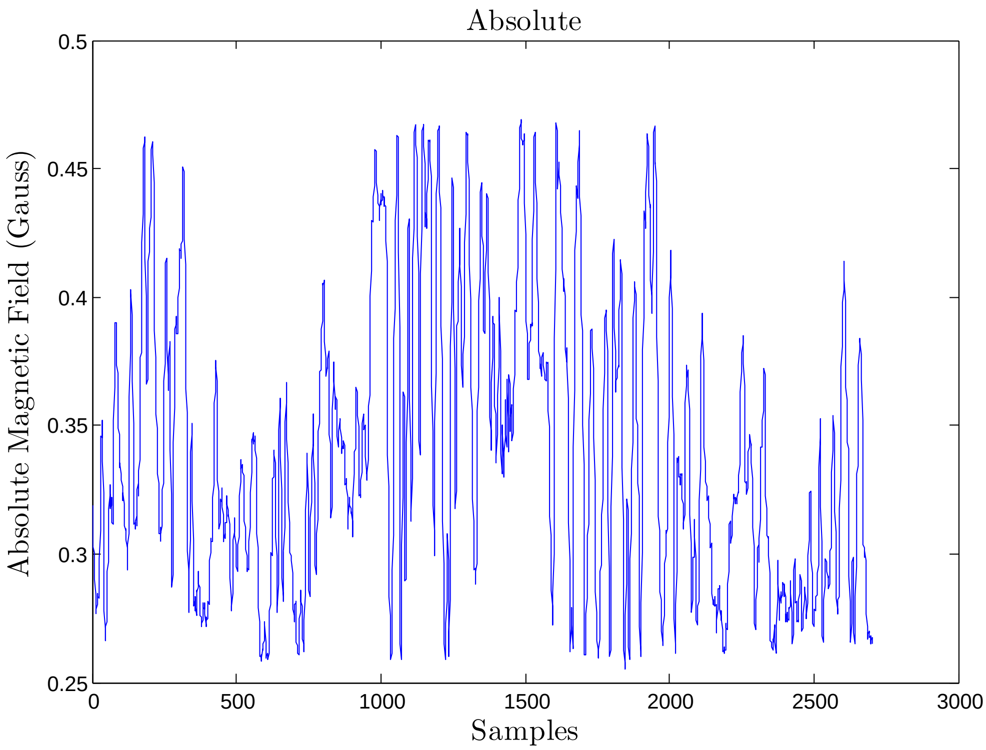
Figure 5.11: The minimization of equation5.3, trying to fit our measurement data to an ellipsoid. The optimization problem was solved fast, due to good initial values.

5.2.2 Calibration test

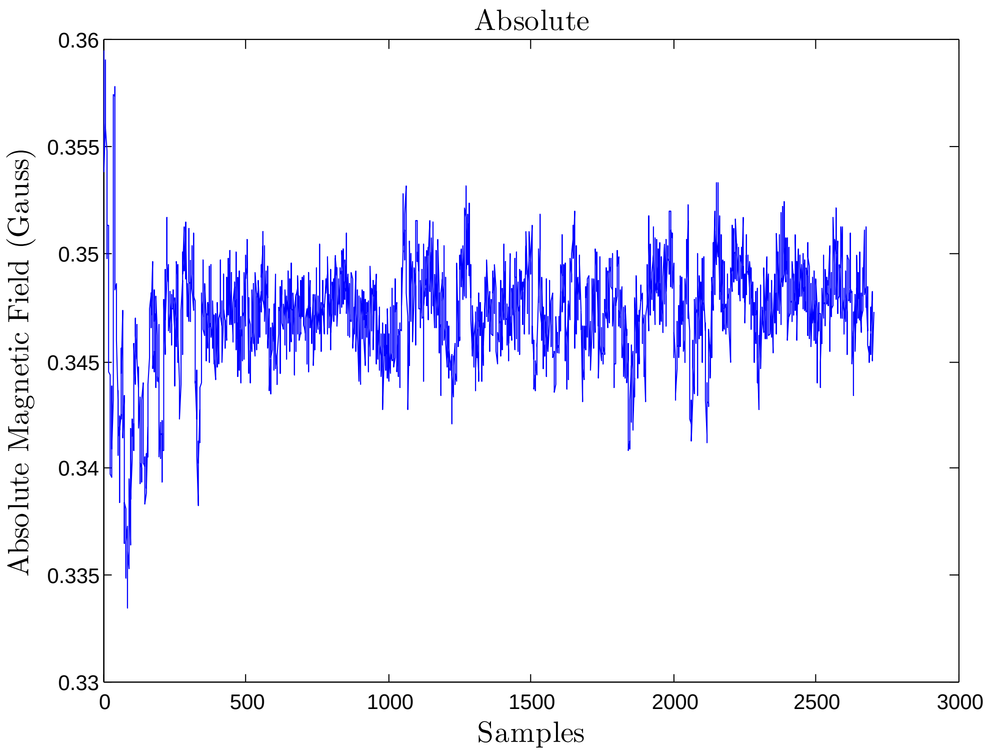
Measurement data as it is plotted in figure 5.13 was obtained by acquiring data through the LabView VI. The ADCS card was simply turned in all arbitrary directions, while the 3D-plot in matlab continuously displayed which areas was measured. In that way, we obtained fairly evenly distributed measurements. In such a test, it is important to be far from any disturbing sources.

5.2.3 Results

The calibration process is simple to perform, and gives a good correction for uneven scaling of the different the axes, bias and misalignment. As long as the sensor is not calibrated to a reference, the accuracy of the magnetic field strength and the rotation between the sensor and the body frame is unknown. Both of these corrections are easily performed by having a good reference. A fluxgate magnetometer is available at the University, and may be a good reference for those corrections.



(a) Uncalibrated



(b) Calibrated

Figure 5.12: Absolute values measured in the HMC5883 magnetometer test.

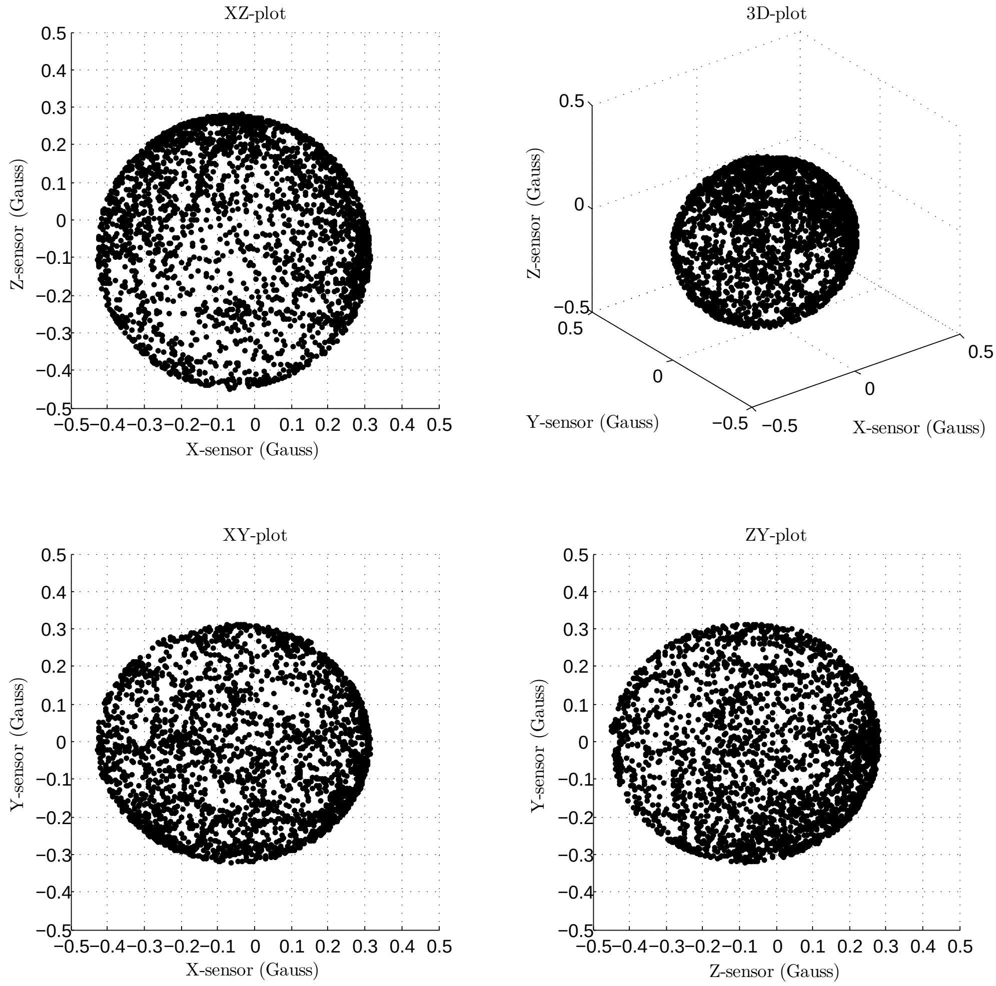


Figure 5.13: Uncalibrated HMC5883L turned in arbitrarily directions inside a building, have generated this data set. We can see that it is weakly elliptical.

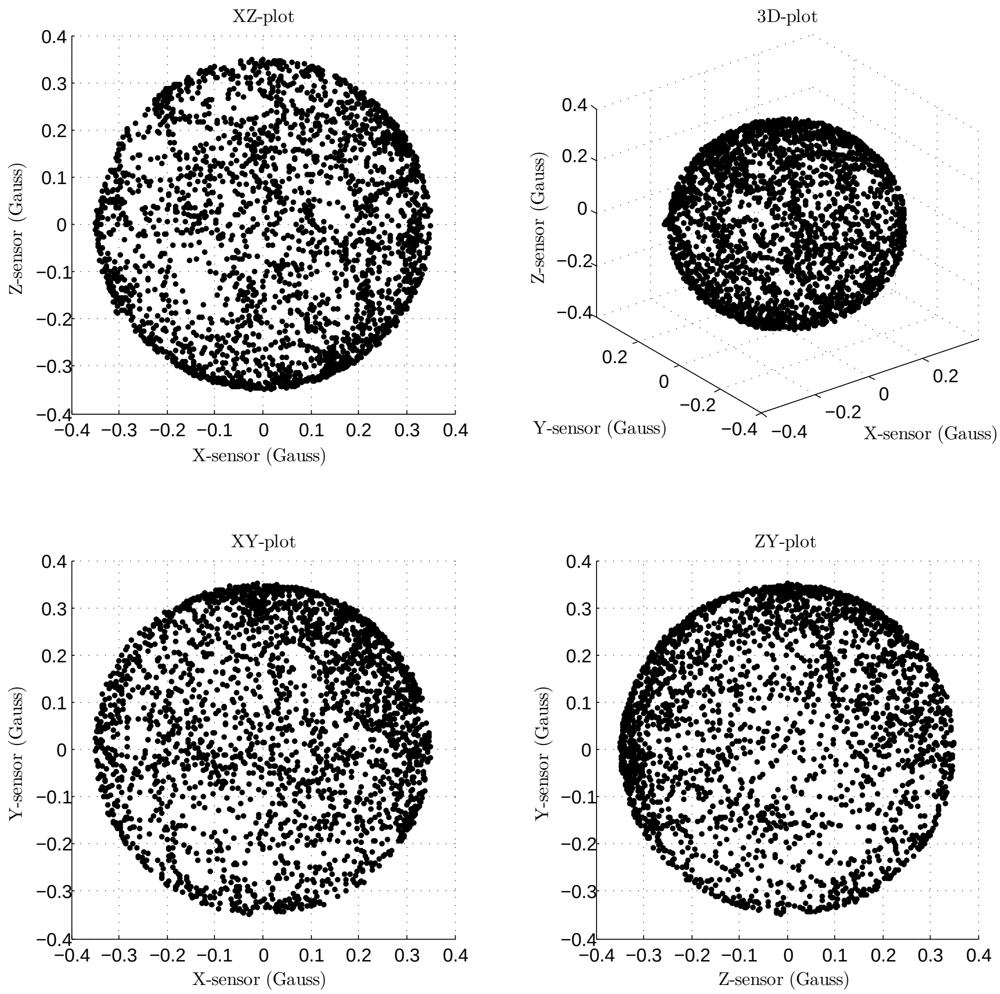


Figure 5.14: Calibrated HMC5883L. The data is corrected after finding the ellipsoid parameters.

Chapter 6

Discussion

The work of this thesis have spanned widely, from mechanical problems, to mathematics and electronics. The following main goals are achieved:

- A first version of the ADCS has been designed, and detumbling is implemented. The ADCS card will be the basis for future work on the ADCS system.
- The firmware for the microcontroller has been programmed, fulfilling its task for a detumbling system, except from the measurement of the magnetorquer current.
- Magnetorquers have been designed, included a method to reproduce new ones, and in different sizes. The Coil winder is successfully tested.
- A LabView VI has been developed in order to perform calibration procedures, and generally control the microcontroller on the ADCS card.
- Two different Gyro sensor setups have been tested, and a calibration method was developed, which should be easy to do again in a later phase of the project. The ITG-3200 was considered as an adequate performance after the calibration.
- A HMC5883L magnetometer is tested, and a calibration method is demonstrated.

6.1 Future Work

The following subjects should be goals for the future work of the

- A current sensing circuit should be implemented for the magnetorquers.
- The magnetometer should be calibrated with an accurate reference.
- The mechanical properties of the satellite should be determined, and the last magnetorquer be produced.
- The hardware platform for the Determination and Control part should be selected, so the next version of an ADCS card can include all the necessary computational power.
- A third sensor should be implemented, this would most likely be a solar sensor, since it is fairly simple, and it has a well defined reference.

- Attitude determination should be developed after, or in the same time as the solar sensor and the hardware is implemented.
- The whole system should be tested thorough, and simulation with hardware in the loop should be performed.

Bibliography

[1] Cal Poly SLO, The CubeSat Program (2009) CubeSat Design Specification Rev. 12 `http://www.cubesat.org/index.php/documents/developers`[1]. 1.1

[2] Bekkeng, T. A. (2009) Prototype Development of a Multi-Needle Langmuir Probe System, Master's thesis, UIO 1.2, 1.2.2

[3] Oredsson, M. (2010) Electrical power system for the CubeSTAR nanosatellite, Master's thesis, UIO 1.2

[4] Stray, F. (2010) Attitude Control of a Nano Satellite, Master's thesis, UIO 1.2, 1.4.1, 2.4.2, 2.4.3, 3.2.1

[5] Tresvig, J. L. (2010) Design of a Prototype Communication System for the CubeSTAR Nano-satellite, Master's thesis, UIO 1.2

[6] Grønstad, M. A. (2010) Implementation of a communication protocol for CubeSTAR, Master's thesis, UIO. 1.2

[7] Vangli, H. (2010) Construction of a remotely operated satellite ground station for low earth orbit communication, Master's thesis, UIO. 1.2

[8] Svartveit, K. (2003) Attitude determination of the NCUBE satellite, Master's thesis, NTNU. 1.4

[9] Krogh, K. and Schreder, Elmo (2002) Attitude Determination for AAU CubeSat, Master's thesis, Aalborg University. 1.4, 2.7

[10] Graversen, T., Frederiksen, M. K. and Vedstesen, S. V. (2002) Attitude Control system for AAU CubeSat, Master's thesis, Aalborg University. 1.4

[11] Wertz, R (1978) Spacecraft Attitude Determination and Control, Kluwer, ISBN 90-277-0959-9. 1.4

[12] Cruise, A. M., Bowles, J. A., Patrick, T. J. and Goodall, C. V. (1998) Principles of Space Instrument Design, Cambridge, ISBN 0-521-45164-7. 1.4

[13] Sidi, M. J. (1997) Spacecraft Dynamics & Control, Cambridge, ISBN 0-521-55072-6. 1.4

[1] Accessed August 2011

[14] Hall, C. D (2003) Spacecraft Attitude Dynamics and Control, www.dept.aoe.vt.edu/~cdhall/courses/aoe4140/ [1]. 1.4, 2.2.3

[15] Seeger, J., Lim, M. and Nasiri, Development of High-Performance, High-Volume Consumer Mems Gyroscopes, InvenSense, www.invensense.com/mems/gyro/documents/whitepapers [1]. 2.5, 2.6

[16] Michael's List of CubeSat Satellite Missions, www.mtech.dk/thomsen/space/cubesat.php [1]. 2.1

[17] Honeywell, Magnetic Sensor Overview, www.magneticsensors.com [1]. 2.3

[18] Honeywell, Application Note AN213, www.magneticsensors.com [1]. 2.2

[19] Fraden, J. (2003) Handbook of Modern Sensors: Physics, Designs, and Applications. 2.2.2

[20] Larson,W. J. and Wertz, J. R., editors (1992) Space Mission Analysis and Design. Microcosm, Inc. and Kluwer Academic Publishers, second edition. 2.2.3

[21] Lerner, L. S. (1995) Physics for scientists and engineers. 3.1, 3.1

[22] Atmel (2007) AVR Butterfly Application Rev07, http://www.atmel.com [1].

[23] Friedel, Jonas and McKibbon, Sean (2011) Thermal Analysis of the CubeSat CP3 Satellite, California Polytechnic State University. 4.2.7.1

[24] Atmel, AN1612 PDI programming driver, www.atmel.com [1]. 4.2.2

[25] Øye, H. K. (2006) Implementering og test av MEMS gyroskop i romfartsapplikasjoner, Master's thesis, UIO. 2.2.2.1, 4.2.3

[26] Sensonor, SAR150 Gyro Sensor Datasheet, www.sensonor.no [1]. 4.3a, 4.3b

[27] InvenSense, ITG-3200 Product Specification, www.invensense.com [1]. 4.5

[28] Honeywell, 3-Axis Digital Compass IC HMC5883L Data Sheet, www.magneticsensors.com [1]. 4.6

[29] Atmel, AVR1003: Using the XMEGA Clock System, www.atmel.com [1]. 4.2.1.1

[30] ROHM, H-bridge drivers Technical Note, www.rohm.com [1]. 4.7, 4.2, 4.8

[31] Kester, W., Bryant, J. and Byrne, M (2008) Grounding Data Converters and Solving the Mystery of "AGND" and "DGND", Tutorial, Analog Devices. 4.2.8

[32] Grødal, A. (1997) Elektromagnetisk kompatibilitet. Tapir forlag. 4.2.8

[33] Atmel, AVR1307: Using the XMEGA USART, www.atmel.com [1]. 4.2.2, 4.4.1

[34] Atmel, AVR1309: Using the XMEGA SPI, www.atmel.com [1]. 4.4.1

[35] Atmel, AVR1308: Using the XMEGA TWI , www.atmel.com [1]. 4.4.1

[36] Bekkeng, J. K. (2007) Prototype Development of a Low-Cost Sounding Rocket Attitude Determination System and an Electric Field Instrument, Article, UiO. 5.1

[37] Bekkeng, J.K (2007) Prototype Development of a Low-Cost Sounding Rocket Attitude Determination System and an Electric Field Instrument, PhD thesis, UiO 5.1

[38] Welch, G. and Bishop, G (2006) An Introduction to the Kalman Filter, University of North Carolina 5.1.4

[39] Brown, R. G. and Hwang, P. Y. C. (1992) Introduction to random signals and applied Kalman filtering 2nd edition, Wiley, ISBN 0-471-52573-1 5.1.4

[40] Vasconcelos, J. F., Elkaim, G., Silvestre, C., Oliveira, P. and Cardeira, B., A Geometric Approach to Strapdown Magnetometer Calibration in Sensor Frame 5.2.1, 5.2.1, 2

Appendix A

Coil Winder User Manual

This appendix is a user guide for the coil winder created in this thesis.

A.1 Overview of Functionality

The coil winder is designed to produce coils in sizes common on CubeSats. The coil core must be printed by a 3D-printer to fit the size and shape desired for the coil. The maximum distance from center of the coil to outer edge (normally corner) of the coil is $120mm$. The maximum thickness is $4mm$, while maximum recommended thickness is $3mm$. The maximum thickness is easily adjusted.

A.2 Understanding the Controller

Control of the coil winder is performed through the potentiometer and the joystick. The potentiometer is used to adjust the rotation speed of the motor, while the joystick is the controller for the rest of the electronics. In A.1, a schematic representation of the menu system is presented. The square boxes represent states the system can be in, while the rounded boxes shows actions which can be performed. The direction of the lines out of each box represents the direction to push the joystick. All values displayed when using the coil winder, is in hundreds of mm. The values describing the position of the guiding wheel, ranging from 0 to 400, is counting from most left position. In other words, 0 to 400 represents the distance $0mm$ to $4mm$ which the servo is pushing the guiding wheel.

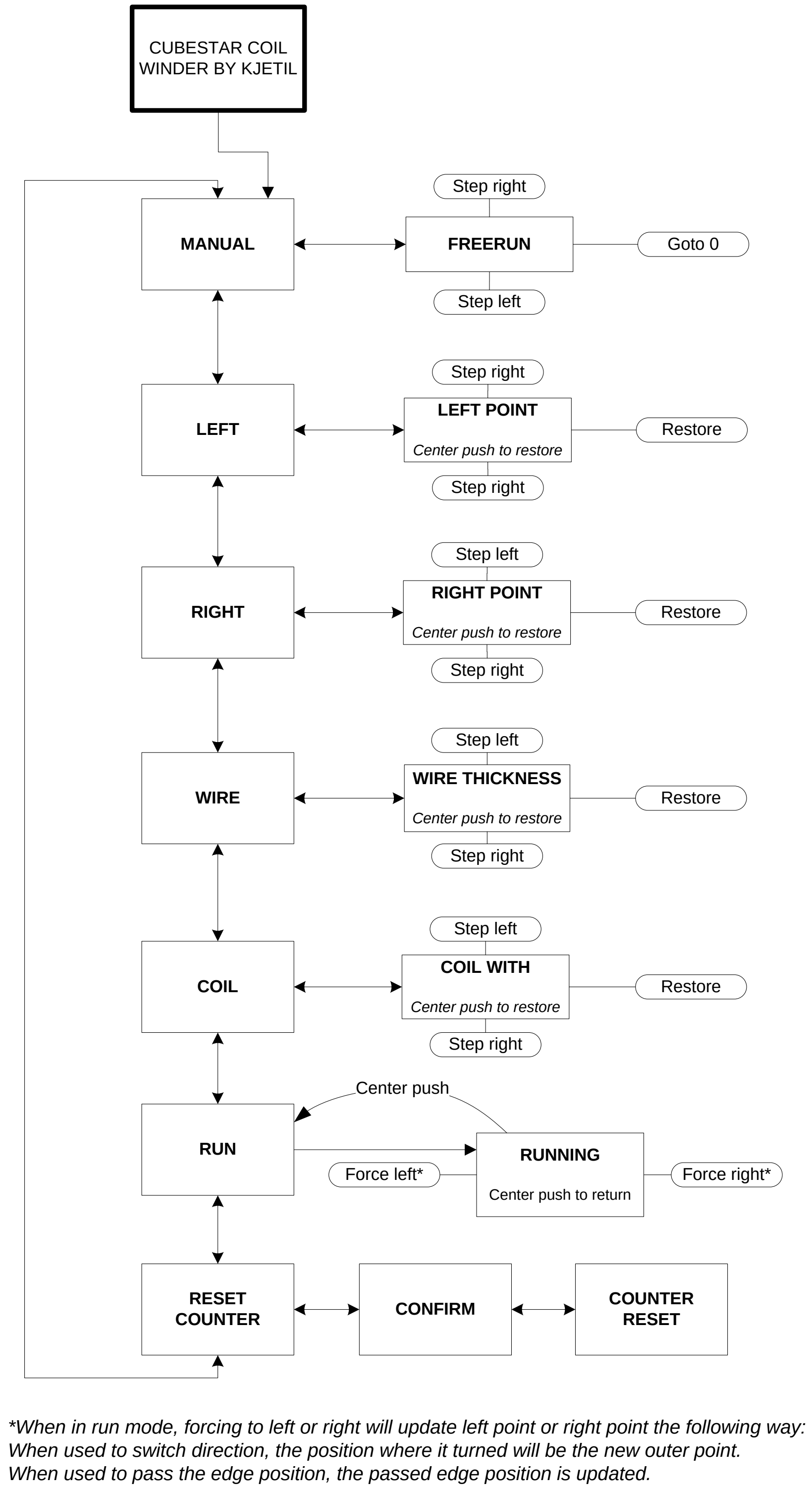


Figure A.1: Flowchart showing the menu system of the coil winder. The direction of the lines and arrows are illustrating the four different directions on the controlling joystick.

A.3 Adhesive and Safety Considerations

Adhesive binding the cobber wires together in the coil frame is required. A low viscosity adhesive is recommended to get the thin wires close to each other. For space applications, low outgassing adhesive is required. A suitable low outgassing epoxy for space purposes are Epoxy Technology U300-2, available from Micro Joining KB in Sweden. U300-2 is a two component low outgassing high viscosity epoxy. Curing time is according to the data sheet$90min$ at $120°C$.

When using EPO-TEK U300-2, the following safety actions should be performed:

- Avoid inhalation. Make sure good ventilation is present when the product is not yet cured.
- The product should not be in contact with human skin.
- Use nitrile protection gloves to avoid contact, do not use latex gloves.
- Use eye protection.

A.4 Step by Step Guide

1. Make sure you have a coil frame with a small hole where you want the ends of the wire to pass through for termination. Remember:
   (a) Two holes can be made, if you want the wire to pass through at different locations.
   (b) The second wire shall be passed through the wire when winded; the winded wires are tighter at the corners and less tight at the middle of the sides. This makes it easier to have the hole close to the middle.
   (c) The holes must be placed in the area of the opening of the plates holding the coil frame.
2. Make sure power is disconnected, so that the servo can be freely rotated.
3. Assemble the coil frame included the shaft holders.
4. By hand, turn the servo wheel in mid position.
5. Mount the assembled coil frame onto the spinning shaft. Try to align it so center of coil frame is directly above the center of the wire guiding wheel.
6. Loosen the wing nuts on the wire break, so you are able to insert a wire.
7. Mount your wire reel at the back shaft.
8. Pass the thread under the back lower shaft, through the wire break, under the front lower shaft, and around the guiding wheel.
9. Put the wire onto the coil frame and passing it through the hole referred to in pt. 1. To temporarily fasten the wire, twist it onto a screw outside the coil frame.
10. A wire working as a "fish tape" should be inserted into the same hole (or the other one, if two separate holes are made). This wire can be twisted into itself on the outside of the coil frame, until the coil is winded up. (Refer to pt. 18 for use).

11. Connect motor, servo and speed controller onto the coil winder card. Connect $12V$ power.
12. Set coil with and wire thickness in the menu system. Refer to A.2.
13. Adjust the setup, by setting left and right edges for where the wire guider should move, and get the right tension of the wire by tighten the wire break. Some notes for adjusting follows:
    (a) To be able to find the right left and right position it might be a good practice to wind the some turns of wire into the coil frame while finding the right values.
    (b) Since the servo are able to push the wire guiding wheel at most $4mm$, the coil frame must be placed directly above, inside these $4mm$. This is best seen while testing, and may be adjusted at this point.
    (c) The left and right point should match the thickness of the coil (right point - left point = coil thickness).
    (d) Wire winded up while testing should be removed, when ready to make the coil. This due to sub-optimal winding while testing.
14. Put the coil frame in upright position with no wire on it, this is the "0 turn" position which the counting starts from.
15. Reset counter.
16. Enter *RUN* mode. Note: The counter is only counting in *RUN* mode.
17. Start the winding process. In the winding process you should do the following:
    (a) Stop winding and add adhesive when necessary. It should always be a layer of liquid adhesive above the wire.
    (b) If the wire stacks up skewed on one of the sides, adjust it with the controller as described in A.2.
    (c) Control the speed, and always keep an eye on the winding.
    (d) Keep an eye on when the coil is full. The wires should not be seen from the side of the coil frame.
18. Remember the start position in pt. 14, and note the stop position. Remember that the counter does not count the last round if you have not fully completed it.
19. Cut the wire.
20. Solder the wire onto the "fish tape" wire, and gently drag it through the hole.
21. Cut, terminate and glue the ends as desired.
22. Disassemble the coil frame.
23. Cure the adhesive as specified in the data sheet[1]. To avoid the coil from bending, cure it under pressure of a flat surface.
24. Inspect and test the coil.
25. Mark the coil with its turn number so you won't forget it.

[1] http://www.epotek.com/sscdocs/datasheets/U300-2.PDF

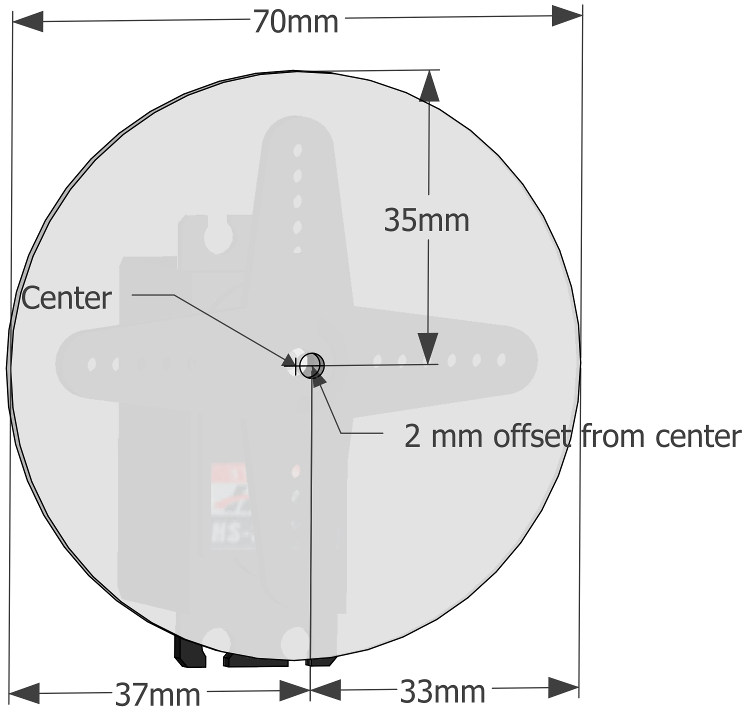


Figure A.2: Circular plastic dish mounted on servo. The distance from rotation axis to left edge in this position is $37mm$. After a 180° rotation, the distance is decreased to $33mm$. The total walk of $4mm$ is changed by applying a new plastic dish with the rotation hole offset changed. The total walk is equal to $offset{*}2$.

A.5 Adjusting Coil Thickness above $3mm$

The maximum coil thickness producible by the coil winder is adjustable by changing the plastic dish mounted on the servo. The present dish with a recommended limit of $3mm$ is actually able to create a $4mm$ thick coil, but some headroom is recommended. It is harder to adjust, and higher uncertainty while using the whole range of a dish. For creating thicker coils than $3mm$, new mounting holes should be made in the plastic dish, or a new dish should be produced. The determining parameter of how thick coil it is possible to wind, is the distance between the center of the plastic dish, and the rotating hole. In figure A.2, the dish which is mounted on the coil winder now, it has a $2mm$ offset, which enables the $4mm$ walk.

Appendix B

Schematics PCB and Part List

B.1 ADCS Card

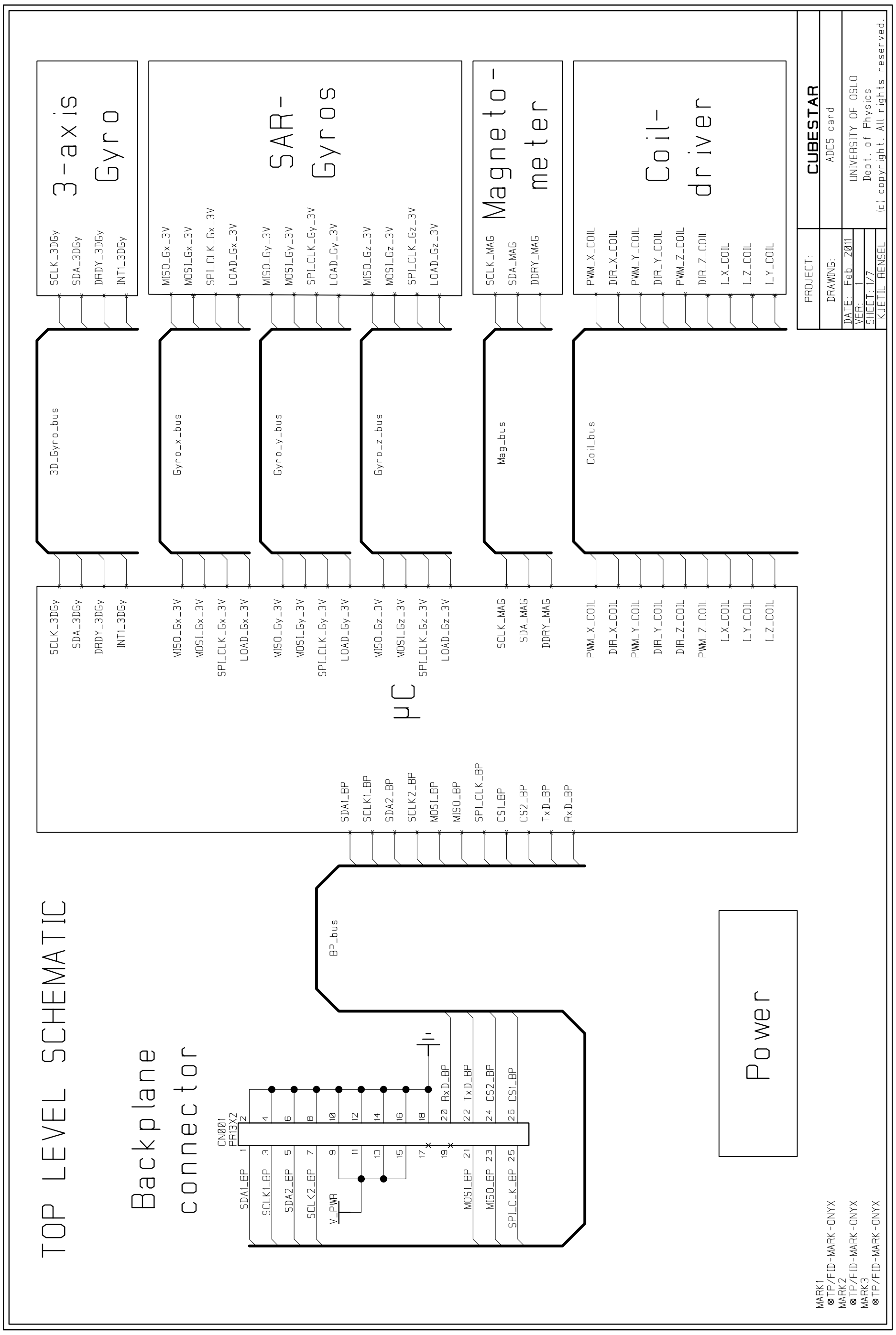


Figure B.1: Schematic ADCS Card, Top Level

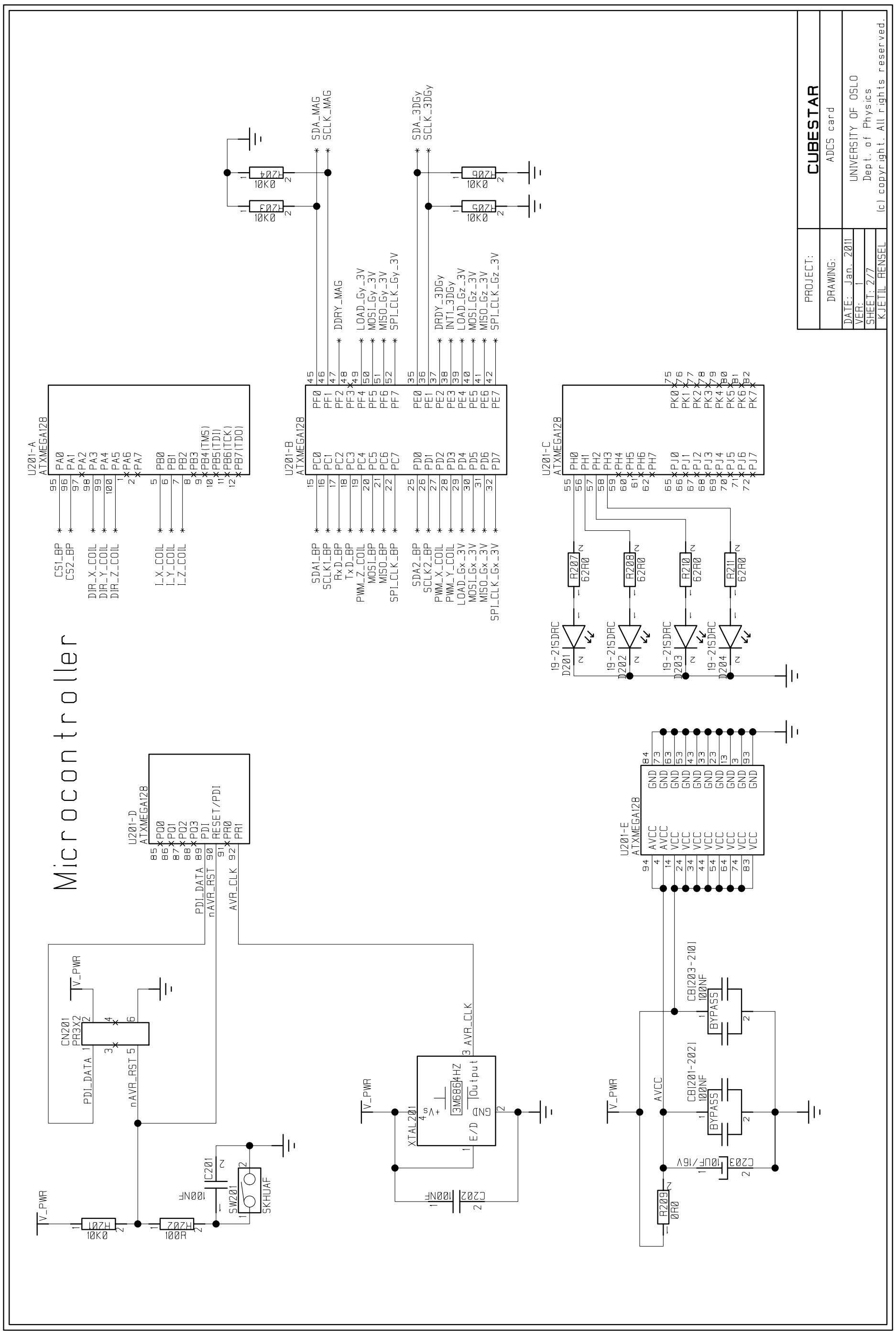


Figure B.2: Schematic ADCS Card, Microcontroller

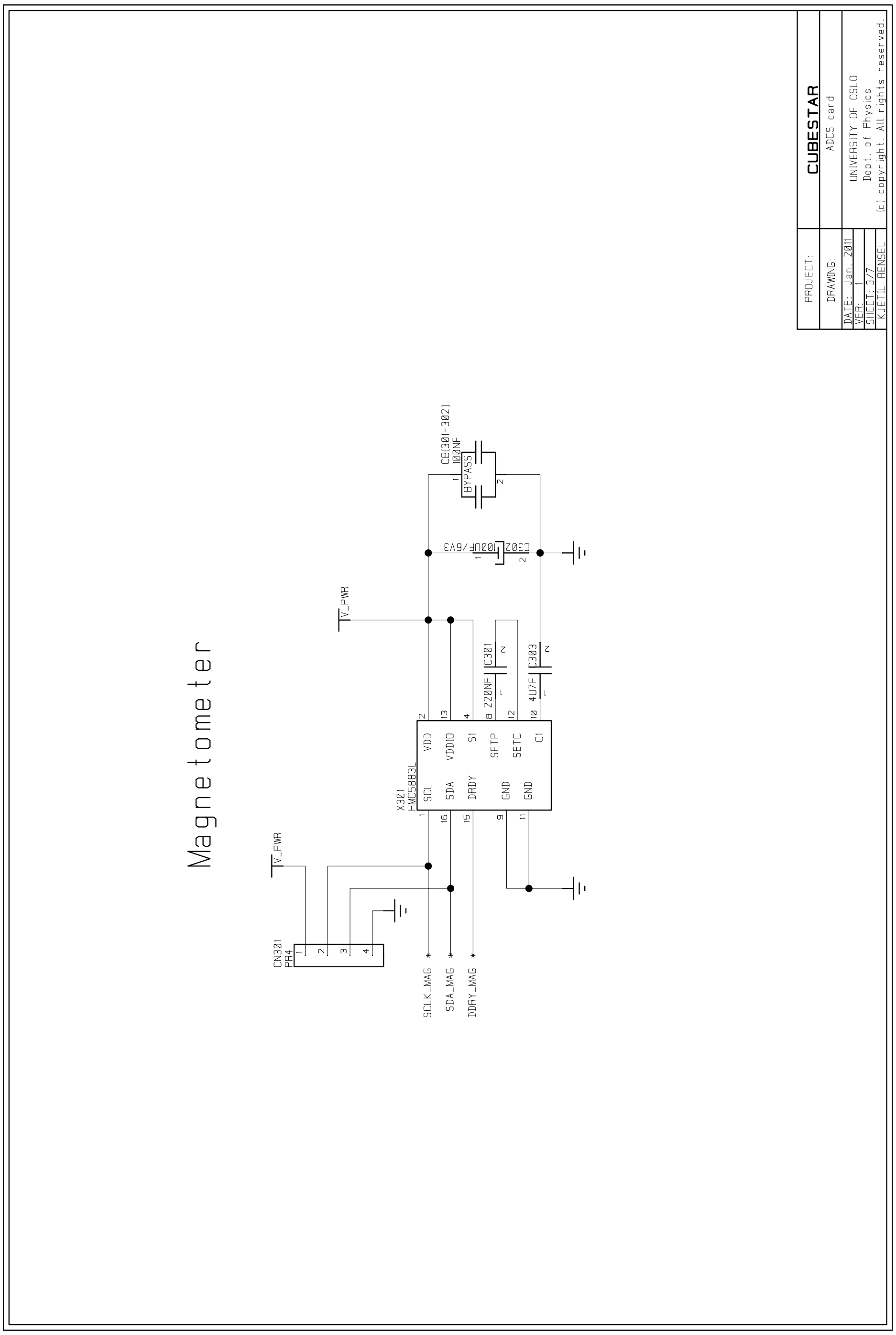


Figure B.3: Schematic ADCS Card, Magnetometer

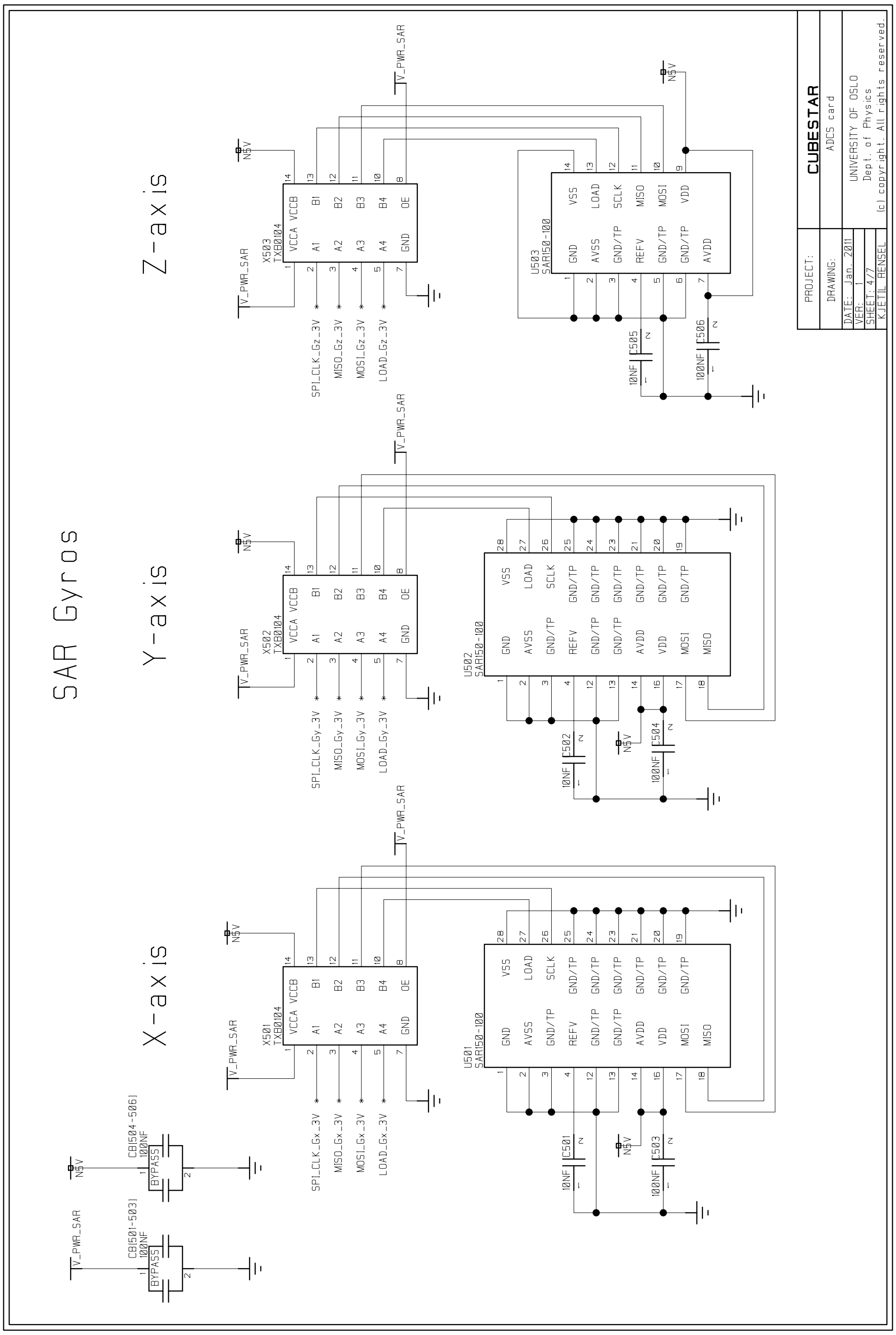


Figure B.4: Schematic ADCS Card, SAR gyros

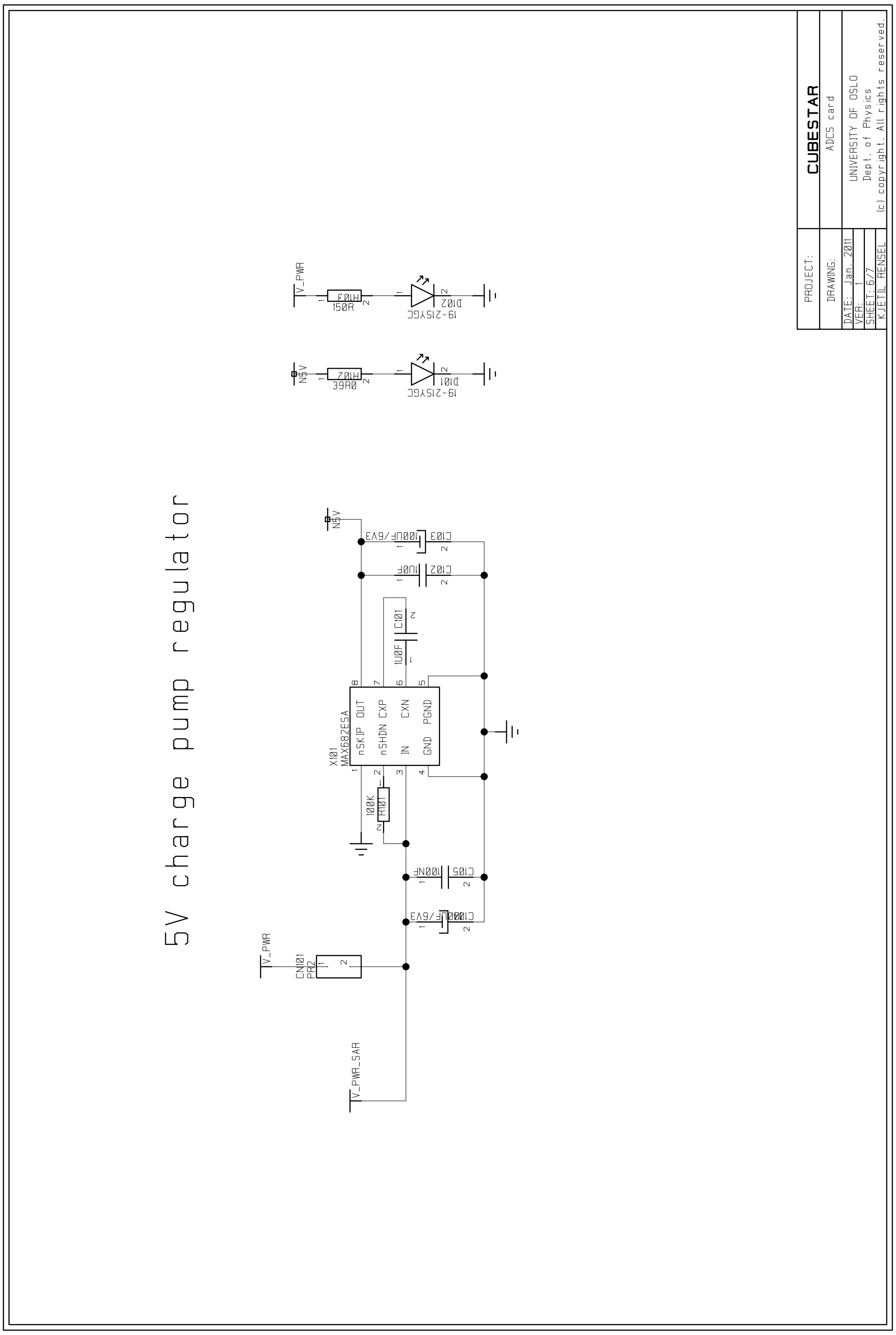


Figure B.5: Schematic ADCS Card, L3G4200D 3-axis gyro

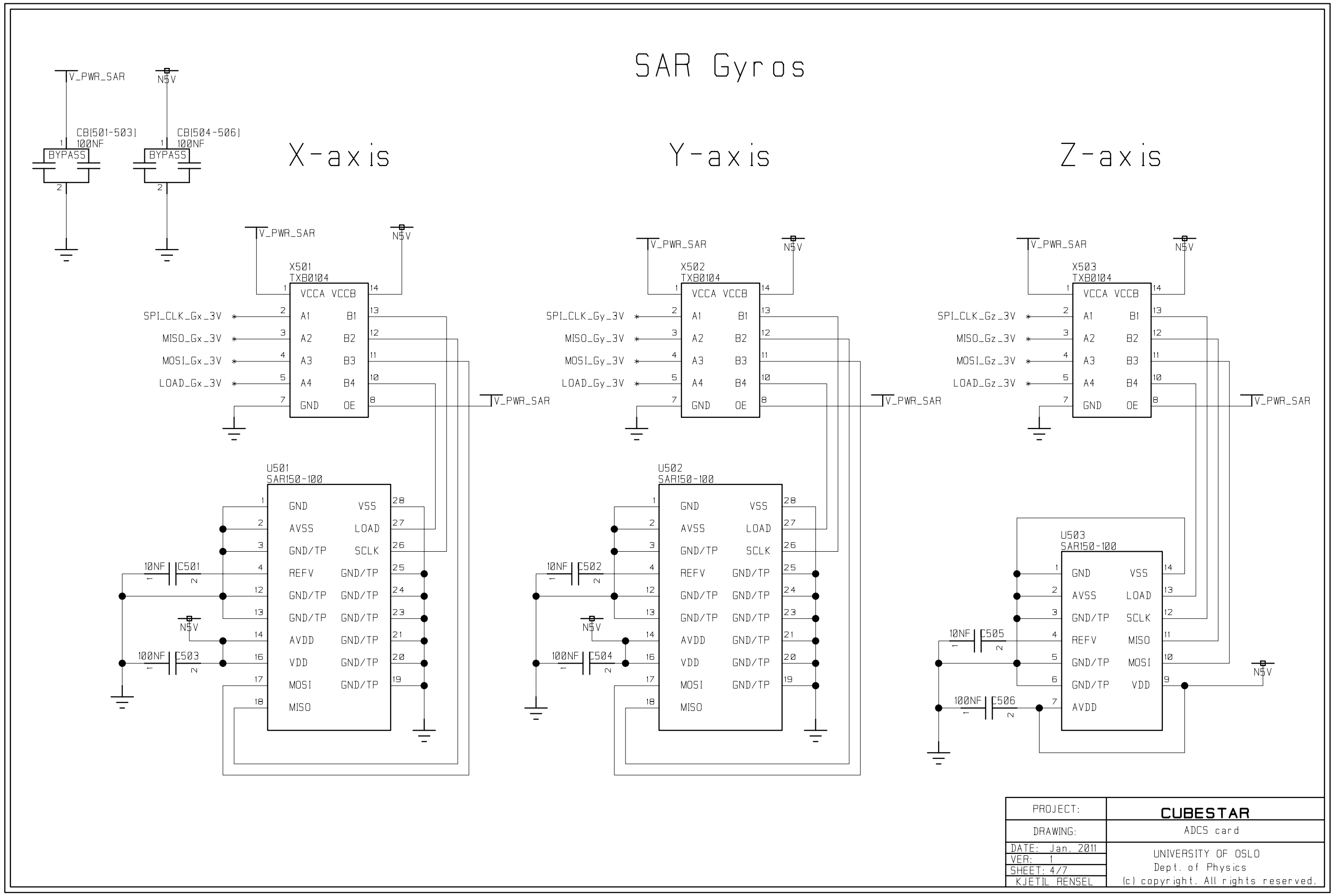


Figure B.6: Schematic ADCS Card, 5V charge pump regulator

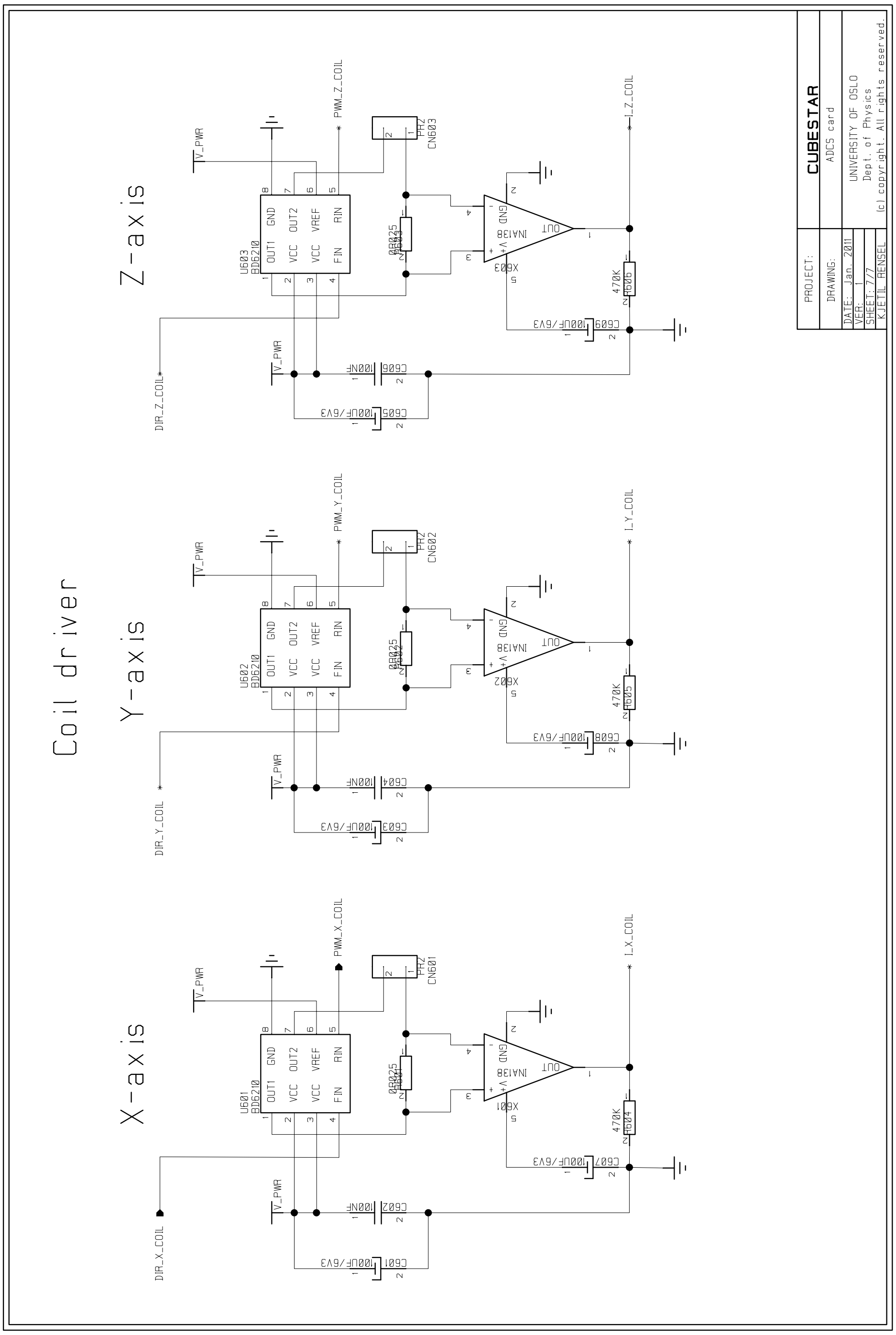
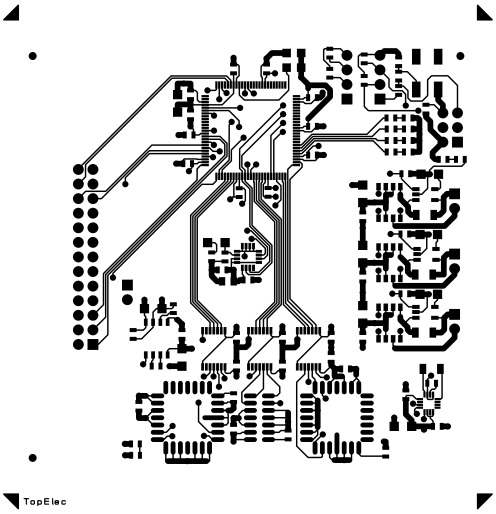
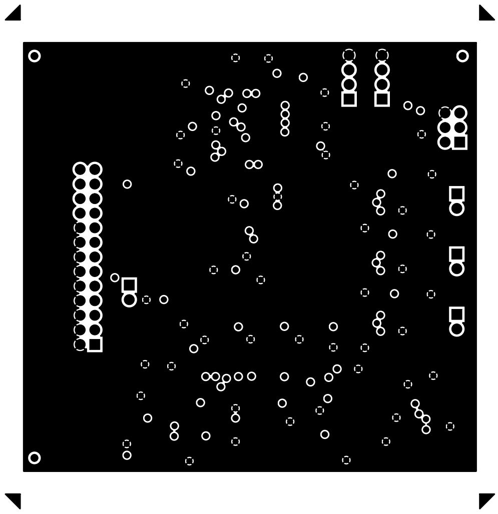


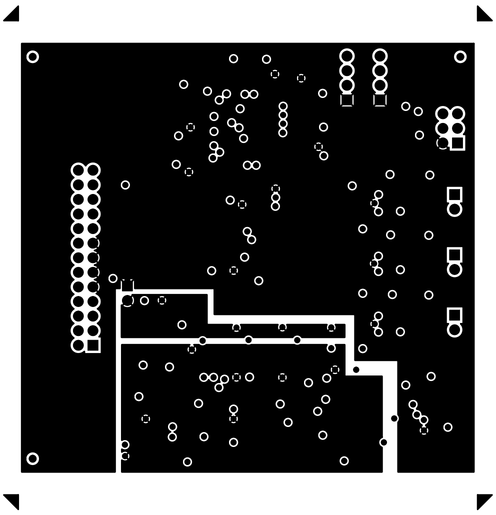
Figure B.7: Schematic ADCS Card, Coil driver



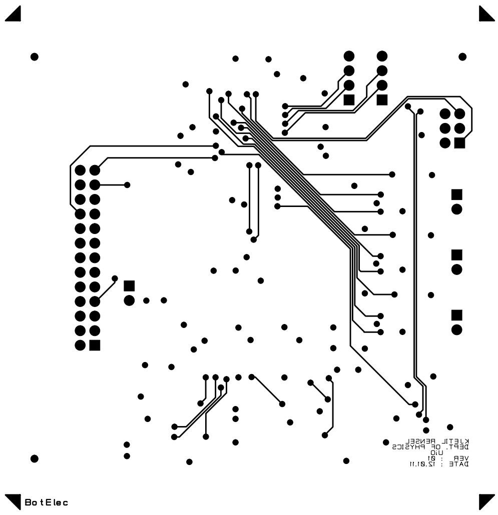
(a) Top Electric



(b) GND



(c) Power



(d) Bottom Electric

Figure B.8: PCB ADCS card

Parts List

CADSTAR Design Editor Version 12.1

Design: M:\Master\Cadstar\ADCS-card\ADCS.pcb

Design Title:

Date: 15. august 2011
Time: 11:03

Part Name	Part Number	Description	Qty.	Comps.
ATMEL/ATXMEGA128/TQFP	X-XX-XXX-XX	ATMEL AVR MICROCONTROLLER	1	U201
BB/INA138/SMD	F-1564888	BURR BROWN CURRENT SHUNT MONITOR	3	X601-603
CAP/100NF/0603R	E-65-759-63	10% 16V 0603 X7R	9	C105
				C201-202
				C503-504
				C506
				C602
				C604
				C606
CAP/10NF/0603R	E-65-758-49	10% 50V 0603 X7R	4	C402
				C501-502
				C505
CAP/10UF/0805R	X-XX-XXX-XX	0805 (Y5V / 10V / E-65-540-67 REEL! 3000) 1 (		
CAP/1U0F/0603R	E-65-202-17	10% 25V 0603 X5R	2	C101-102
CAP/220NF/1206R	E-65-777-04	20% 50V 1206 X7R	1	C301
CAP/4U7F/0603R	F-1833806	AVX 10% 10V 0603 X5R	1	C303
CAP/BYPASS/0603R	E-65-759-63	10% 16V 0603 X7R	20	CB201-210
				CB301-302
				CB401-402
				CB501-506
CON/PR13X2PIN/HORIZ	E-43-714-31	13X2 TYCO PINROW ANGELED	1	CN001
CON/PR2	E-43-702-19	2 SCOTT ELEC. PINROW	4	CN101
				CN601-603
CON/PR3X2	E-43-704-33	3X2 SCOTT ELEC. PINROW	1	CN201
CON/PR4	E-43-702-19	4 SCOTT ELEC. PINROW	2	CN301
				CN401
LED/19-21SDRC/SMD	E-75-308-01	SMD LED RED	4	D201-204
LED/19-21SYGC/SMD	E-75-312-47	SMD LED GREEN	2	D101-102
MAXIM/MAX682ESA	F-1380017	3.3V-INPUT TO REGULATED 5V-OUTPUT, CHARGE PUMPS	1	
RES/0R00/0603R	E-60-440-02	RESISTOR KOA 0603 1% 0.1W	1	R209
RES/0R025/1206R	F-1703806	CURRENT SENSE RESISTOR 1206 1% 0.25W	3	R601-60
RES/100K/0603R	E-60-452-64	RESISTOR KOA 0603 1% 0.1W	1	R101
RES/100R/0603R	E-60-445-49	RESISTOR KOA 0603 1% 0.1W	1	R202
RES/10K0/0603R	E-60-450-25	RESISTOR KOA 0603 1% 0.1W	6	R201
				R203-206
				R401
RES/150R/0603R	E-60-445-80	RESISTOR KOA 0603 1% 0.1W	1	R103
RES/39R0/0603R	E-60-444-40	RESISTOR KOA 0603 1% 0.1W	1	R102
RES/470K/0603R	E-60-454-21	RESISTOR KOA 0603 1% 0.1W	3	R604-606
RES/62R0/0603R	E-60-444-99	RESISTOR KOA 0603 1% 0.1W	4	R207-208
				R210-211
SPES/BD6210	F-1716258	H-BRIDGE DRIVER	3	U601-603
SPES/HMC5883L/SMD	X-XX-XXX-XX	3-AXIS DIGITAL COMPASS IC	1	X301
SPES/L3G4200D	X-XX-XXX-XX	MEMS MOTION SENSOR: 3-AXIS DIGITAL OUTPUT GYROSCOPE		
SPES/SAR150-100/HOR	X-XX-XXX-XX	SENSONOR GYRO SENSOR. HIGH PREC. HORISONTAL MOUNT. RAT		
SPES/SAR150-100/VER	X-XX-XXX-XX	SENSONOR GYRO SENSOR. HIGH PREC. VERTICAL MOUNT. RATE		
SPES/TXB0104PWR	F-1607891	4-BIT BIDIRECTIONAL VOLTAGE-LEVEL TRANSLATOR	3	
SW/SKHUAF/SMD	E-35-790-18	ALPS-SMD PUSH BUTTON	1	SW201
TANT/0U47F/25V/SMD	E-67-737-41	KEMET T491 TANTAL ELECTROLYTIC CAP	1	C403
TANT/100UF/6V3/SMD	F-1135257	AVX TANTAL ELECTROLYTIC CAP	9	C103-104
				C302
				C601
				C603
				C605
				C607-609
TANT/10UF/16V/SMD-A	E-67-702-83	KEMET T491 SERIES 20%	1	C203
XTAL/3M6864HZ/CFPS-69	F-1276668	IQD CFPS-69 +/- 50PPM SMD L.P OSC	1	XTAL201

End of report

Figure B.9: Part List ADCS card

B.2 Mini Backplane Card

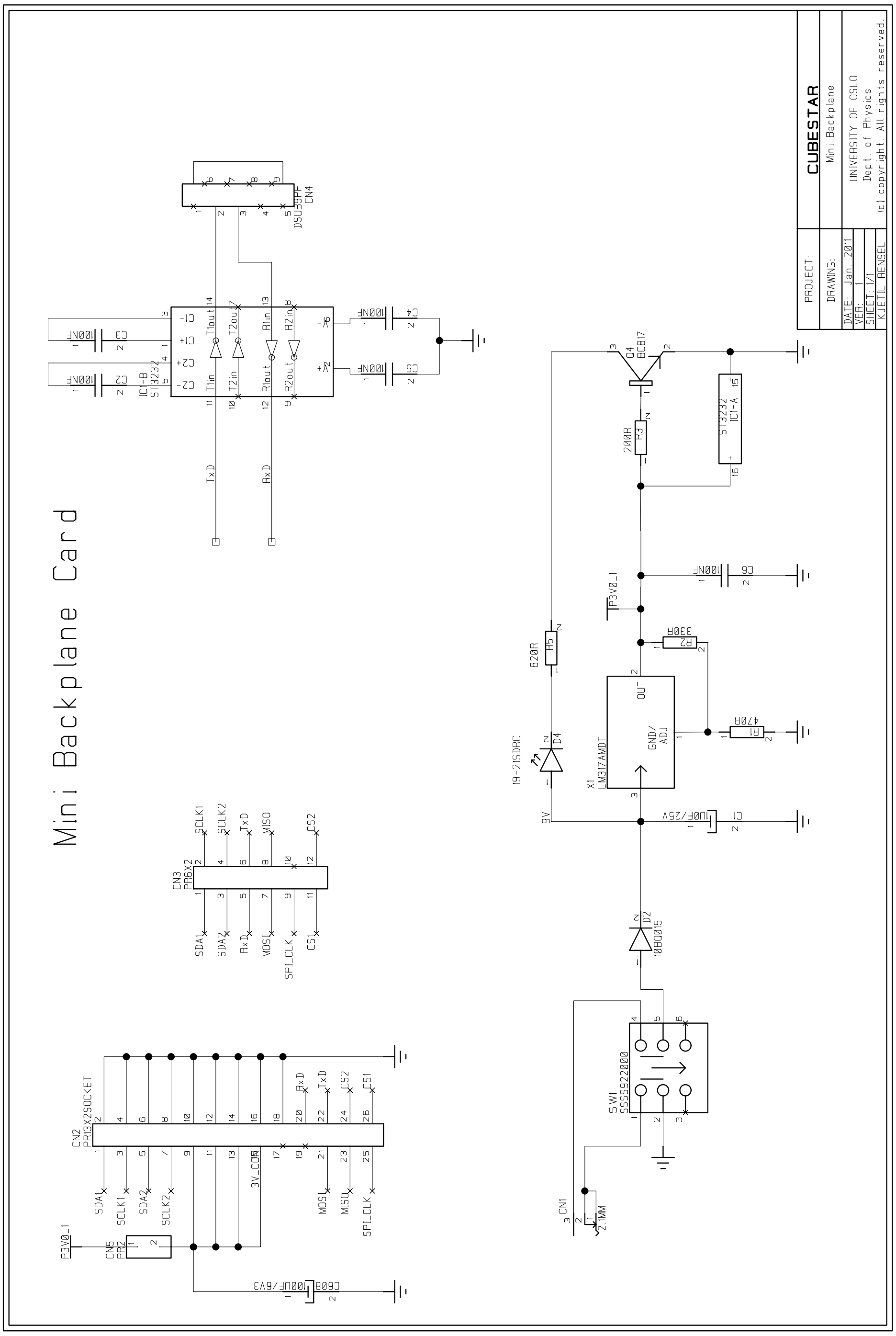
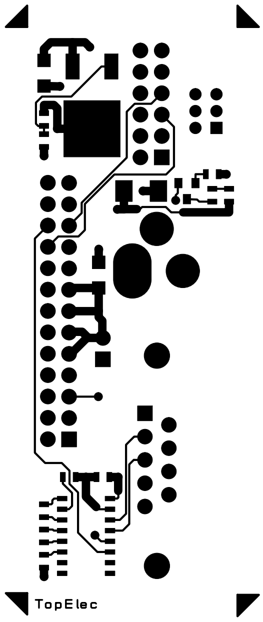
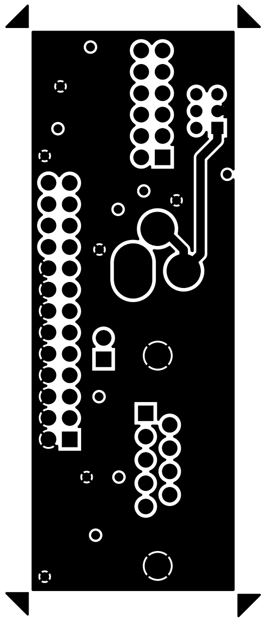


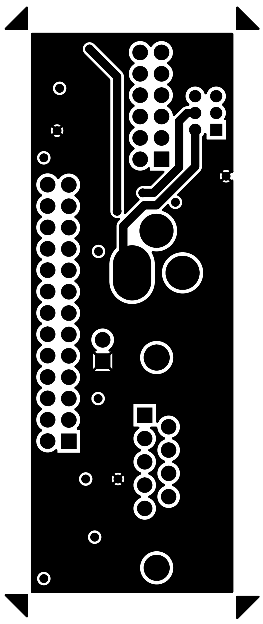
Figure B.10: Schematic Mini Backplane Card



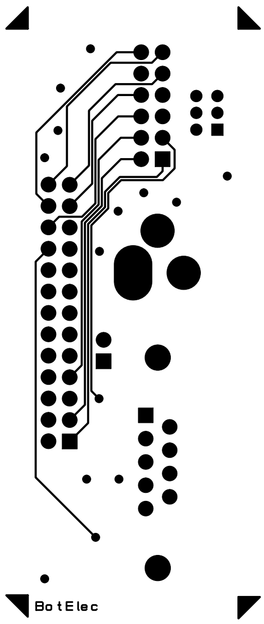
(a) Top Electric



(b) GND



(c) Power



(d) Bottom Electric

Figure B.11: PCB Mini Backplane Card

```
------------------------------------------------------------------------------
                                   Parts List

                       CADSTAR Design Editor Version 12.1

Design:     M:\Master\Cadstar\MiniBackplane\MiniBackplane.pcb

Design Title:


Date:       15. august 2011
Time:       11:01

------------------------------------------------------------------------------
Part Name              Part Number            Description                      Qty.   Comps.
---------              -----------            -----------                      ----   ------

CAP/100NF/0603R        E-65-759-63            10% 16V 0603 X7R                 5      C2-6
CON/DSUB9PF_GND/MM     E-44-057-00            9PIN ANGLED DSUB-CON FEMALE (2.84mm)  1       CN4
CON/ELMCH2/            E-42-051-59            BAT.ELEM.CONNECTOR 2.1 MM        1      CN1
CON/PR13X2SOCKET/VER   E-43-782-12            13X2 TYCO SOCKET                 1      CN2
CON/PR2                E-43-702-19            2 SCOTT ELEC. PINROW             1      CN5
CON/PR6X2              E-43-704-33            6X2 SCOTT ELEC. PINROW           1      CN3
DIO/10BQ015            E-70-217-02            IR. SMD -VERY LOW DROP SCHOTTKY DIODE 15V/1A   1
LED/19-21SDRC/SMD      E-75-308-01            SMD LED RED                      1      D4
POW/LM317AMDT          E-73-266-56            ADJ. POS. REGULATOR TO-252       1      X1
RES/200R/0603R         E-60-446-14            RESISTOR KOA 0603 1% 0.1W        1      R3
RES/330R/0603R         E-60-446-63            RESISTOR KOA 0603 1% 0.1W        1      R2
RES/470R/0603R         E-60-447-05            RESISTOR KOA 0603 1% 0.1W        1      R1
RES/820R/0603R         E-60-447-62            RESISTOR KOA 0603 1% 0.1W        1      R5
ST/ST3232CD/SMD-S      E-73-217-48            DUAL RS-232 TX/RX 3.0V - 5.5V    1      IC1
SW/SSSS922000          E-35-111-36            ALPS SLIDE SW.                   1      SW1
TANT/100UF/6V3/SMD     F-1135257              AVX TANTAL ELECTROLYTIC CAP      1      C608
TANT/1U0F/25V/A        E-67-713-64            TANTAL ELECTROLYTIC CAP          1      C1
TRAN/BC817/SMD         E-71-006-39            NPN TRANSISTOR SOT23 45V/0.5A 0.25W.   1      Q4
------------------------------------------------------------------------------
                                 End of report
------------------------------------------------------------------------------
```

Figure B.12: Part List Mini Backplane Card

B.3 Coil Winder Card

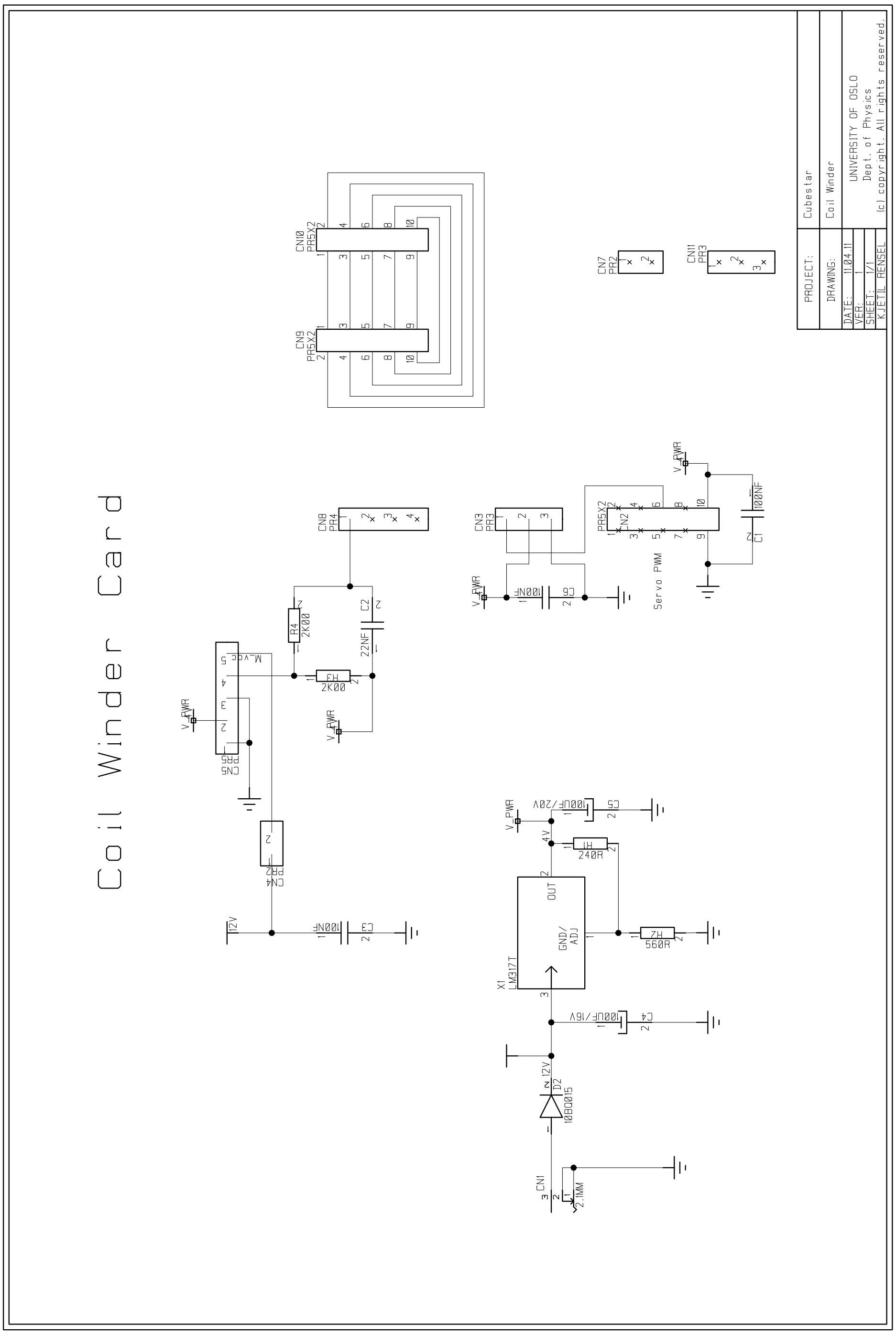
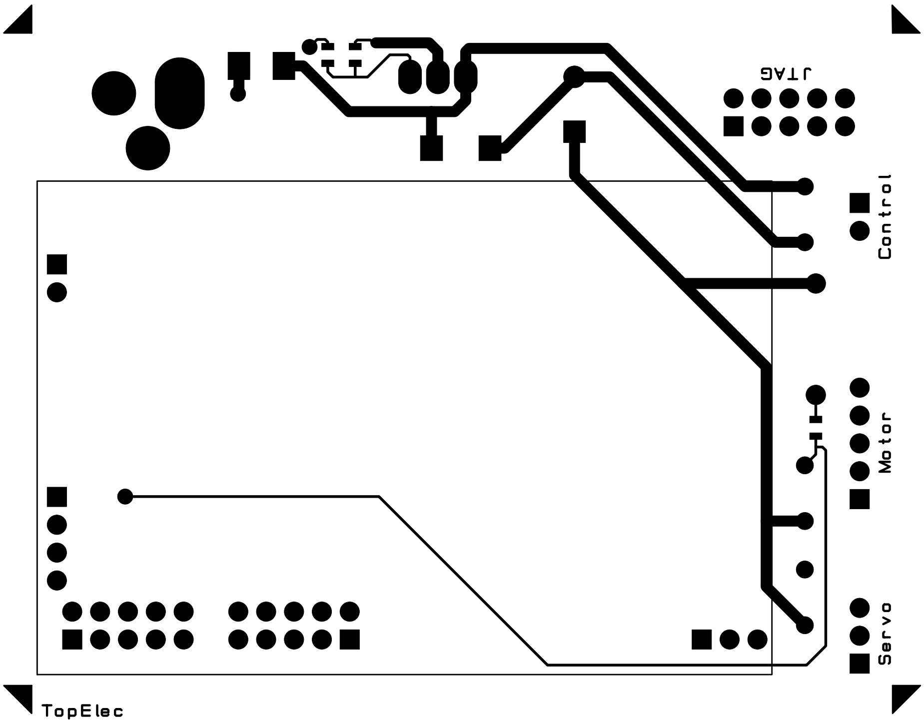
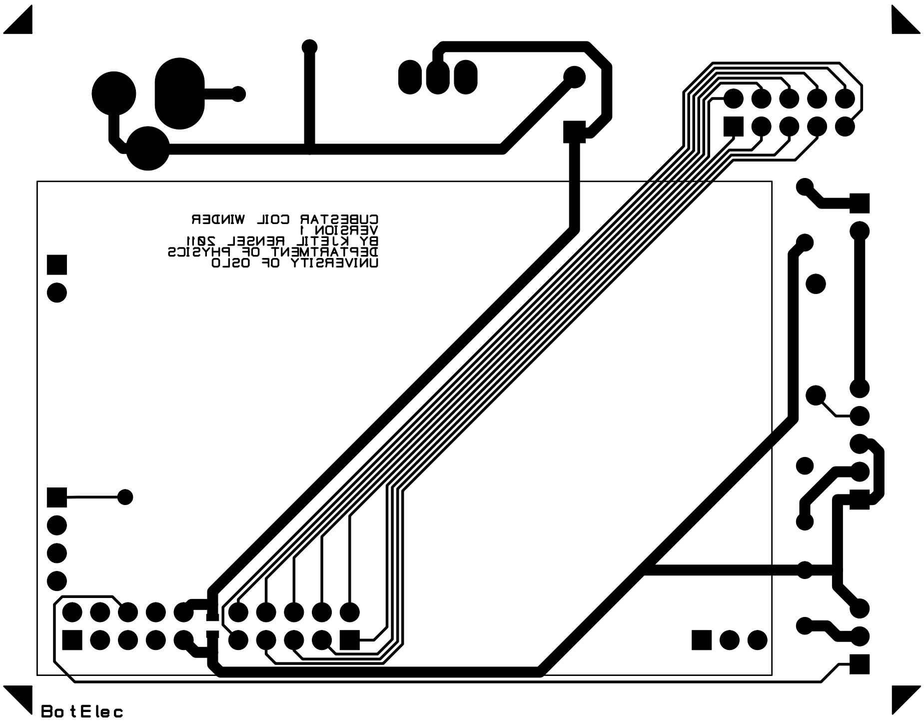


Figure B.13: Schematic Coil Winder Card



(a) Top Electric



(b) Bottom Electric

Figure B.14: PCB Coil Winder Card

```
-------------------------------------------------------------------------------
                                   Parts List

                       CADSTAR Design Editor Version 12.1

Design:      M:\Master\Cadstar\CoilCard\Coil.pcb

Design Title:
ELAB-2011

Date:       15. august 2011
Time:       11:02

-------------------------------------------------------------------------------
Part Name              Part Number            Description                      Qty.   Comps.
---------              -----------            -----------                      ----   ------

CAP/100NF/0603R        E-65-759-63            CERAMIC CAP X7R +/-10%  16V      1      C1
CAP/100NF/CER          E-65-736-87            KEMET CK05BX  50V (5mm SP)       2      C3
                                                                                      C6
CAP/22NF/CER           E-65-735-88            KEMET CK05BX  50V (5mm SP)       1      C2
CON/ELMCH2/            E-42-051-59            BAT.ELEM.CONNECTOR 2.1 MM        1      CN1
CON/PR2                E-43-702-19            2 SCOTT ELEC. PINROW             2      CN4
                                                                                      CN7
CON/PR3                E-43-702-19            3 SCOTT ELEC. PINROW             2      CN3
                                                                                      CN11
CON/PR4                E-43-702-19            4 SCOTT ELEC. PINROW             1      CN8
CON/PR5                E-43-702-19            5 SCOTT ELEC. PINROW             1      CN5
CON/PR5X2              E-43-704-33            5X2 SCOTT ELEC. PINROW           3      CN2
                                                                                      CN9-10
DIO/10BQ015            E-70-217-02            IR. SMD -VERY LOW DROP SCHOTTKY DIODE 15V/1A   1
POW/LM317T/TO220       E-73-120-77            ADJ. POS. REGULATOR TO-220       1      X1
RES/240R/0603R         E-60-446-22            RESISTOR KOA 0603 1% 0.1W        1      R1
RES/2K00/0603R         E-60-448-53            RESISTOR KOA 0603 1% 0.1W        1      R4
RES/2K00/0W6           E-60-726-07            FIRSTRONICS RM0207S 1% 0.6W      1      R3
RES/560R/0603R         E-60-447-21            RESISTOR KOA 0603 1% 0.1W        1      R2
TANT/100UF/16V/C       F-1793885              TANTAL ELECTROLYTIC CAP          1      C4
TANT/100UF/20V/RAD     E-67-200-64            SANYO SA/SC 20% ELYT             1      C5
-------------------------------------------------------------------------------
                                 End of report
-------------------------------------------------------------------------------
```

Figure B.15: Part List Coil Winder Card

Appendix C

LabView Source Code

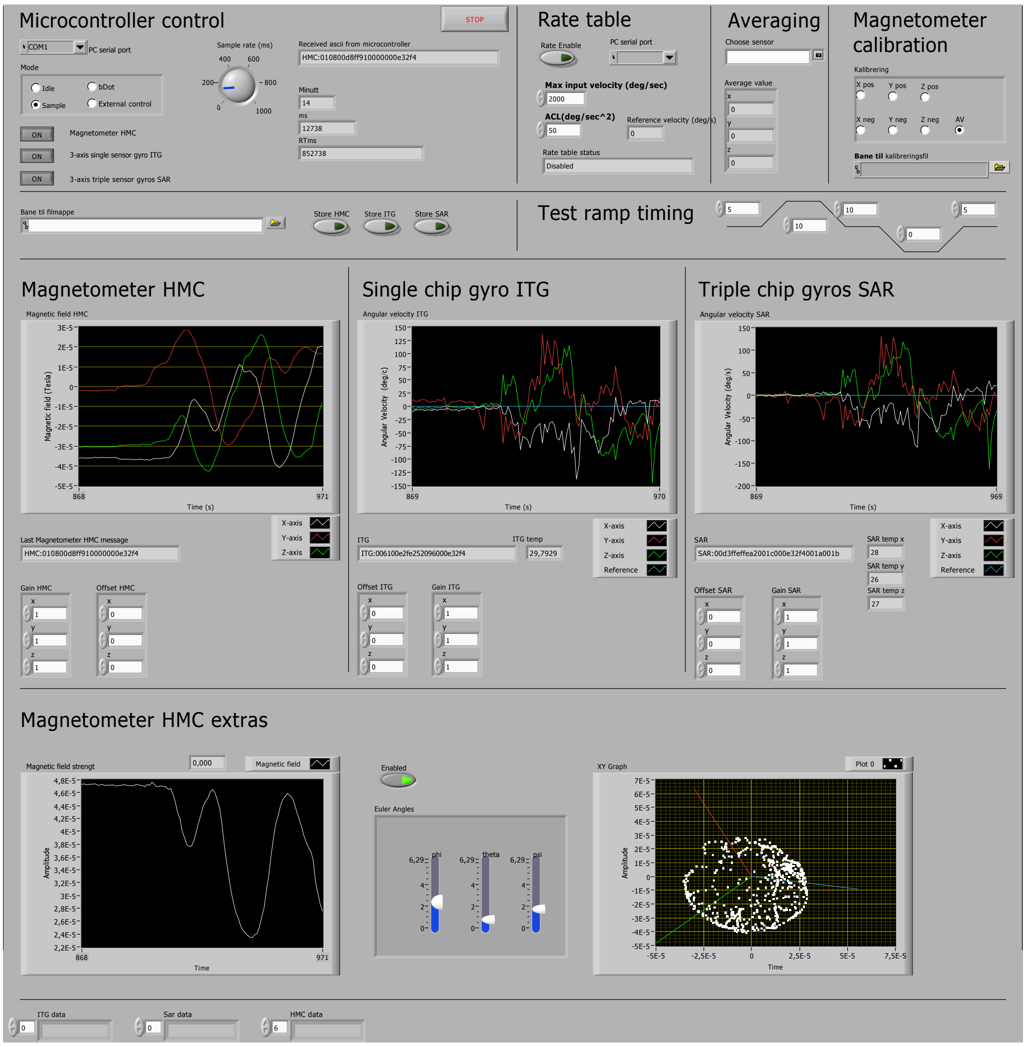


Figure C.1: Front Panel of the LabView VI ADCSmate.vi

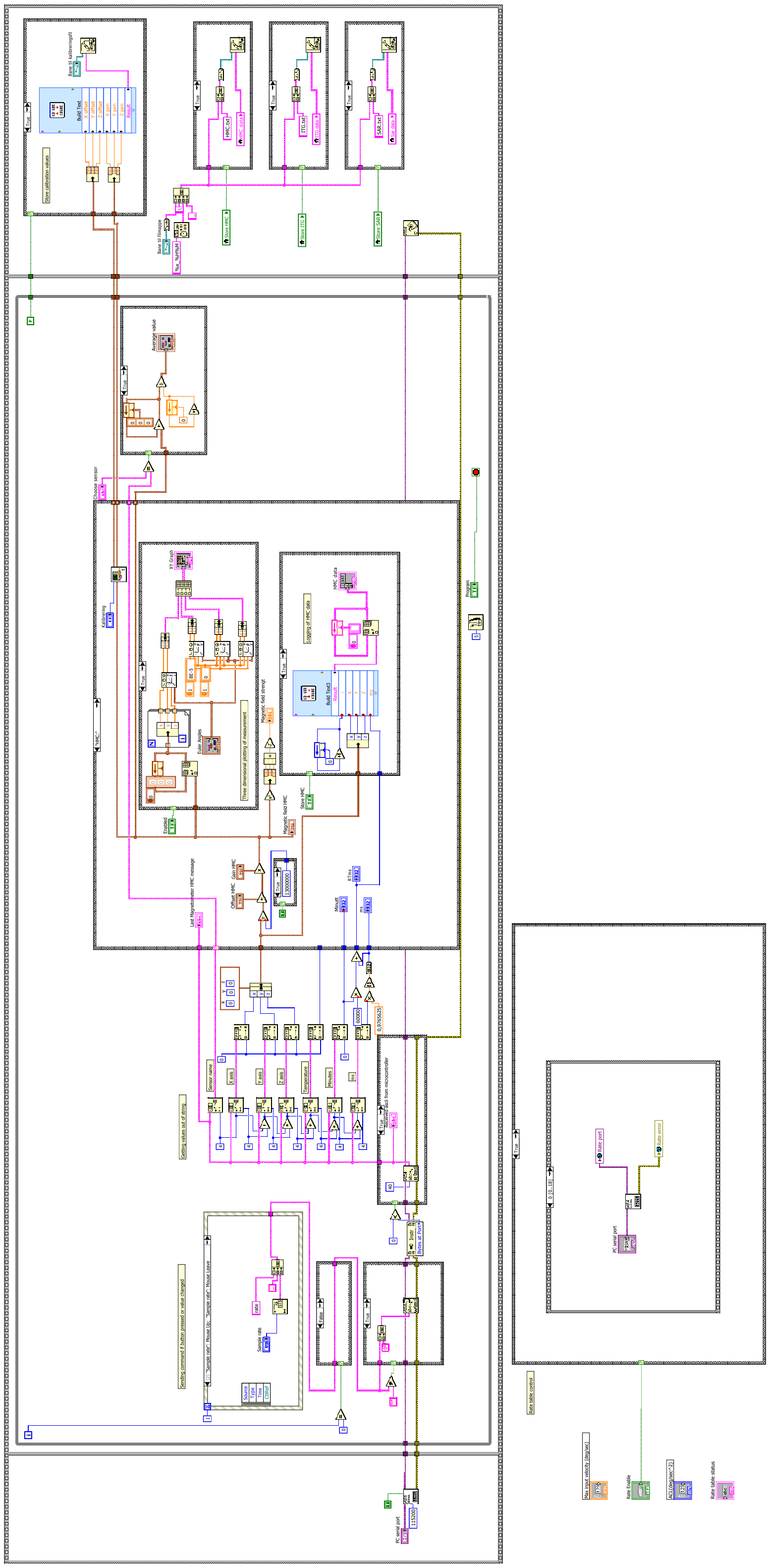


Figure C.2: Block Diagram, Complete with rate table controller

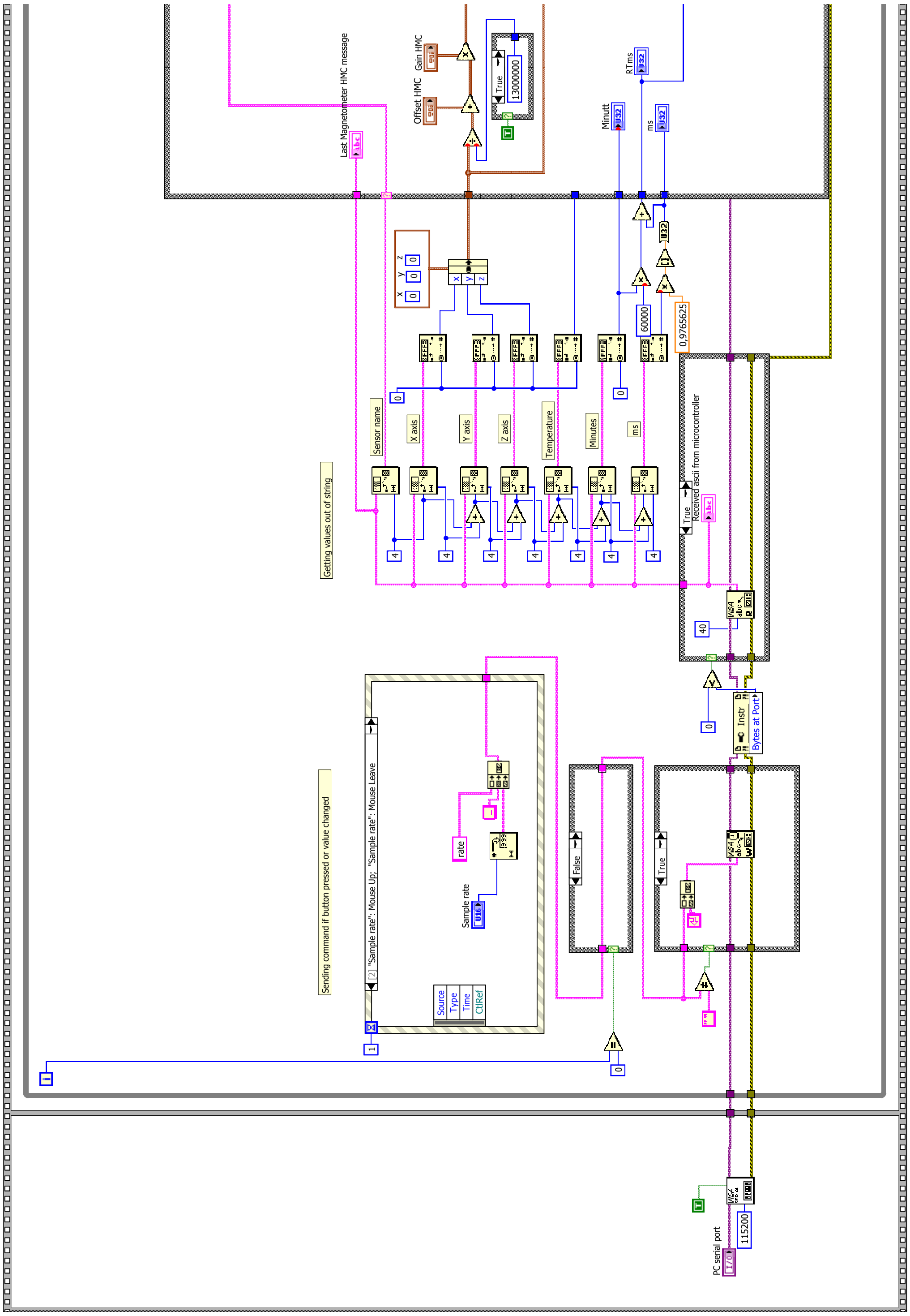


Figure C.3: Block Diagram, main left of the LabView VI ADCSmate.vi

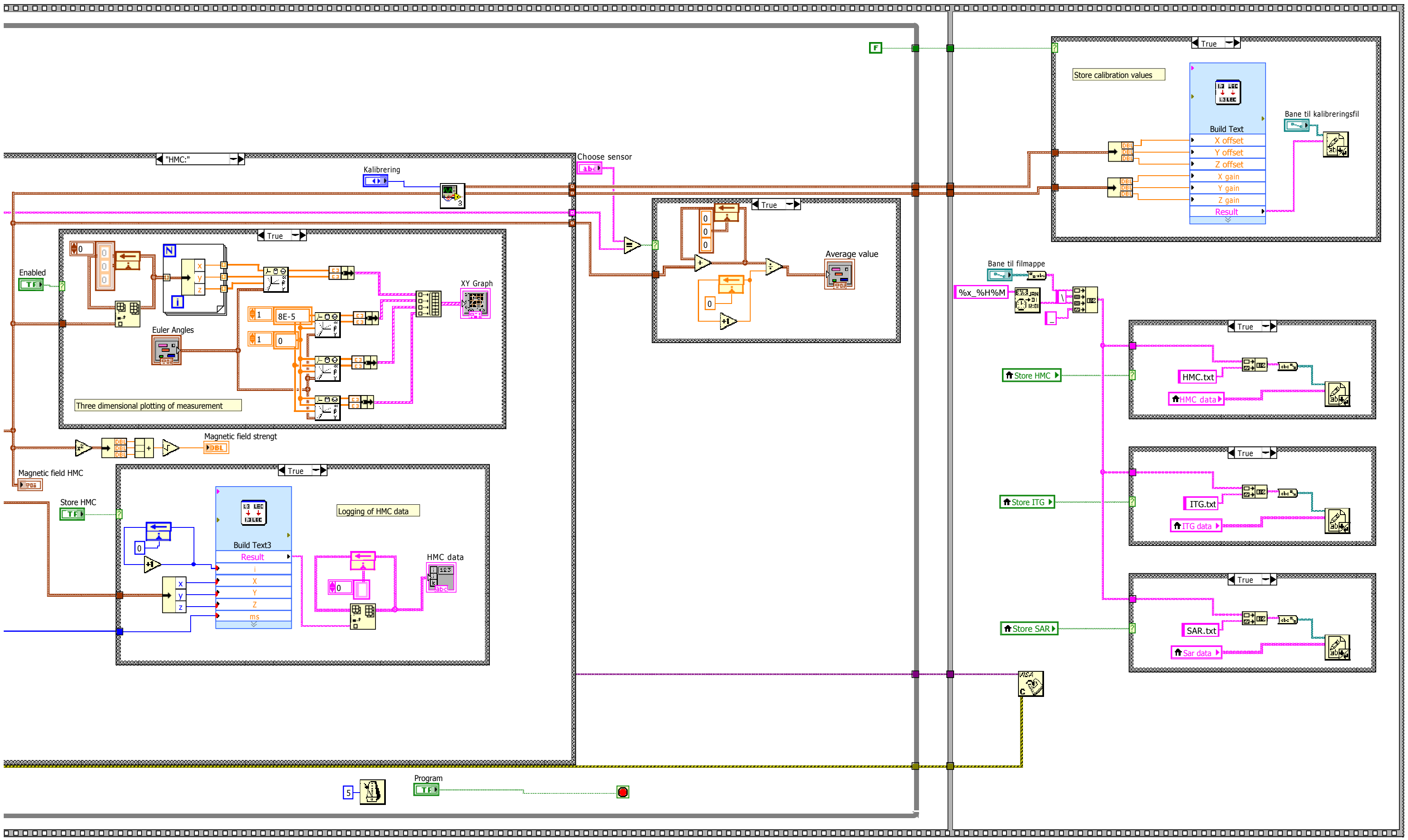


Figure C.4: Block Diagram, main right of the LabView VI ADCSmate.vi

Appendix D

Microcontroller Source Code

D.1 ADCS Card

Listing D.1: main.h

```
#define arraysize(ar)  (sizeof(ar) / sizeof(ar[0]))

/*! Selects the Usart */
#define USART USARTC0

/*! Defining number of bytes in buffer. */
 #define NUM_BYTES        16

/*! BAUDRATE 100kHz and Baudrate Register Settings */
#define BAUDRATE  100000
#define TWI_BAUDSETTING TWI_BAUD(F_CPU, BAUDRATE)

typedef struct magnetorquer {
  int16_t desiredValue;
  int16_t actualCurrent;
  uint8_t temperatureFactor;
  volatile uint16_t *forwardOutput;
  volatile uint16_t *reverseOutput;
  bool direction;
} magnetorquer_t;

typedef struct realTimeClock {
  uint8_t minute;
  uint8_t hour;
  uint8_t day;
  uint8_t year;
} realTimeClock_t;

typedef enum stateMachine {
  st_start,
  st_executeCommand,
  st_sleep,
  st_sample,
  st_externalControl,
  st_print,
  st_bDot,
  st_activateMagnetorquer
} state_t;
```

```

typedef struct
{
    state_t state;
    state_t (*pFunc)(void);
} menu_state_t;


static int uart_putchar (char c, FILE *stream);
void init_clk();
void init_uart();
void init_hmc5883();
void init_itg3200();
void init_sar150();
void init_SampleTimer(TC1_t * timer, uint16_t compare);
void init_RTC32();
void init_coils();
void init_adc();

state_t f_start(void);
state_t f_executeCommand(void);
state_t f_sleep(void);
state_t f_sample(void);
state_t f_externalControl(void);
state_t f_print(void);
state_t f_bDot(void);
state_t f_activateMagnetorquer(void);



menu_state_t menu_state[] = {
//   STATE                              STATE_FUNC

  {st_start                    ,     f_start                },
  {st_executeCommand           ,     f_executeCommand       },
  {st_sleep                    ,     f_sleep                },
  {st_sample                   ,     f_sample               },
  {st_externalControl          ,     f_externalControl      },
  {st_print                    ,     f_print                },
  {st_bDot                     ,     f_bDot                 },
  {st_activateMagnetorquer     ,     f_activateMagnetorquer },

  {0                           ,     NULL}
};
```

Listing D.2: main.c

```
/*******************************************************
*
*  File:    main.c
*  Project: CubeSTAR ADCS card version 1
*  Author:  Kjetil Rensel
*  Revised: August 2011
*
*******************************************************/

#define F_CPU 3.6864E6

/***********************  INCLUDES  ***********************/
#include <avr/io.h>
#include <stdlib.h>
```

```
#include <stdio.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <float.h>

#include "spi_driver.h"
#include "twi_master_driver.h"
#include "usart_driver.h"
#include "HMC5883.h"
#include "SAR150.h"
#include "itg3200.h"
#include "main.h"

/*********************  DEFINITIONS  *********************/
//Coil control PWM period: 3686400Hz/0x7F=29026Hz => OK
#define pwmPeriod 0x007F
#define adcMuxNeg0AndPos4_gc 0x00;
#define adcMuxNeg1AndPos5_gc 0x09;
#define adcMuxNeg2AndPos6_gc 0x12;
#define SAMPLE_TIMER  &TCE1

#define coilCurrent 30

#define xAxis 0
#define yAxis 1
#define zAxis 2
#define threeAxis 3

/***********************  MACROS  ***********************/
#define TC_SetPeriod( _tc, _period ) ( (_tc)->PER = (_period) )
#define TC_Reset( _tc ) ( (_tc)->CNT = 0 )
#define getTimerMs( _tc) ((uint16_t)(((_tc)->CNT)/3.6))
#define CounterPeriod 3600 //F_CPU/prescaler = 3.6864E6/1024
#define crossProduct1(_a, _b) ((float)(_a)[1]*(float)(_b)[2] - (float)(_a)[2]*(float)(_b)[1])
#define crossProduct2(_a, _b) ((float)(_a)[2]*(float)(_b)[0] - (float)(_a)[0]*(float)(_b)[2])
#define crossProduct3(_a, _b) ((float)(_a)[0]*(float)(_b)[1] - (float)(_a)[1]*(float)(_b)[0])
#define sendStatus() printf("STA:%05d-%1d-%1d-%1d-%1d\r\n", *SAMPLE_TIMER.PER, mode, sampleHmc, sampleItg, sampleSar)

/**********************  VARIABLES  **********************/
  // Sensor calibration values
  // Negative scale value to correct for mounting the sensors
  // the opposite way in that particular axis.

typedef struct sensorCalibrationParameters
  {
    int16_t bias[3];
    float scale[3];
    float misalignment[3][3];
  } sensorCalibrationParameters_t;


sensorCalibrationParameters_t    hmcCalibtationParameter =
  {{ 0,  0,  0},   //bias  {x, y, z}
   { 1,  1,  1},   //scale {x, y, z}
   {               //misal.     x    y    z
   { 0,  0,  0},   //        x { 0 , xy, xz}
```

```
    { 0,   0,   0},   //       y { yx, 0 , yz}
    { 0,   0,   0}}}; //       z { zx, zy, 0 }

sensorCalibrationParameters_t    itgCalibtationParameter =
   // {{-102, 129,  23},//bias  {x, y, z}
   {{  0, 0,  0},   //bias  {x, y, z}
    { -1,  -1,  1}, //scale {x, y, z}
    {               //misal.     x   y   z
    { 0,   0,   0},   //       x { 0 , xy, xz}
    { 0,   0,   0},   //       y { yx, 0 , yz}
    { 0,   0,   0}}}; //       z { zx, zy, 0 }

sensorCalibrationParameters_t    sarCalibtationParameter =
   {{ 0,   0,   0},   //bias  {x, y, z}
    { 1,  -1,  -1}, //scale {x, y, z}
    {               //misal.     x   y   z
    { 0,   0,   0},   //       x { 0 , xy, xz}
    { 0,   0,   0},   //       y { yx, 0 , yz}
    { 0,   0,   0}}}; //       z { zx, zy, 0 }


 // Hardware variables, data structure definition found
 // in corresponding driver file
realTimeClock_t       realTimeClock;
USART_data_t          USART_data;
TWI_Master_t          twiHmc;
hmc_Measurement_t     hmcMeasurement[threeAxis];
TWI_Master_t          twiItg;
itgMeasurement_t      itgMeasurement;
SPI_Master_t          spiSar[threeAxis];
SPI_DataPacket_t      spiDataPacketSar[threeAxis];
sarMeasurement_t      sarMeasurement[threeAxis];
magnetorquer_t        magnetorquer[threeAxis];

/* bDotFactor is:
 * Magnetic Moment / mA: 3.69
 * itgScale (1/14.375)
 * hmcScale (1/1300)
 * sarScale 0.1  (Not in use)
 * bDotGain -1 (Tesla: -10000, Gauss: -1) */

 static float bDotFactor = - 3.69 * pwmPeriod / 14.375 / 1300;

//Definition of state variables
state_t state;

bool sampleTimerCompareMatch = false;
bool sampleItg = false;
bool sampleHmc = false;
bool sampleSar =false;
bool bDotSample = false;
bool commandReceived = false;

enum modes {
  idle,
  sample,
  bDot,
  externalControl
} mode = idle;

//Set up UART stdout
```

```
static FILE mystdout = FDEV_SETUP_STREAM (uart_putchar, NULL, ↩
    _FDEV_SETUP_WRITE);

/*********************** FUNCTIONS **********************/

static int uart_putchar (char c, FILE *stream) {
/*A link between stdout and USART driver.
 *
 * Sends a char if software buffer is not full.
 * The function must have this exact argumets and return values
 * to work with stdout. Only char c is utilized.
 */



  //whaits for free space in buffer;
  while (USART_data.buffer.TX_Tail - USART_data.buffer.TX_Head == 1);
    //Sends byte to software buffer.
    USART_TXBuffer_PutByte(&USART_data, c);
  return 0;
}


// *** Initialization functions***

void init_clk() {
// Processor clock initialization

   /* To activate external clock, the following must be done:
     1. Select external clock as source in XOSCCTRL (External osc ctrl ↩
         register)
     2. Enable with XOSCEN (External osc enable) in OSC.CTRL
     3. Whait for external clock to be stable.
     4. Enable change in CLK.CTRL by write right value to CCP
     5. Select external clock as main clock source
  */

  OSC.XOSCCTRL |= OSC_XOSCSEL_EXTCLK_gc;         // 1
  OSC.CTRL |= OSC_XOSCEN_bm;                     // 2
  while ( (OSC.STATUS & OSC_XOSCRDY_bm) == 0 ); // 3
  CCP = CCP_IOREG_gc;                            // 4
  CLK.CTRL = CLK_SCLKSEL_XOSC_gc;                // 5
}

void init_uart() {
// UART initialization

  /*  Setting up UART using atmels device drivers:
    1. Set output and input pin
    2. Use USART defined and initialize buffers. Sets interruptlevel
    3. Use USART defined, set 8 Data bits, No Parity, 1 Stop bit.
    4. Enable RXC interrupt.
    5. Set Baudrate to 115200 bps, values calculated by ATMELs ↩
        Baudrate_calculations.xls spreadsheet
    6. Whait for baudrate to take effect (strange character is outputted ↩
        if not performed)
    7. Enable both RX and TX.
  */

  PORTC.DIRSET   = PIN3_bm;             // 1 TX
  PORTC.DIRCLR   = PIN2_bm;             // 1 RX
```

```
  USART_InterruptDriver_Initialize(&USART_data, &USART, USART_DREINTLVL_LO_gc);         // 2
  USART_Format_Set(USART_data.usart, USART_CHSIZE_8BIT_gc, USART_PMODE_DISABLED_gc, false); // 3
  USART_RxdInterruptLevel_Set(USART_data.usart, USART_RXCINTLVL_LO_gc);                  // 4
  USART_Baudrate_Set(&USART, 1 , 0);     // 5
  _delay_ms(10);                          // 6
  USART_Tx_Enable(USART_data.usart);     // 7
  USART_Rx_Enable(USART_data.usart);     // 7
}

void init_hmc5883() {
// HMC5883 magnetometer and TWI PORTF initialization

  //TWIF
    // Initiate TWI F, utilizing library.
    // Setting up F0 and F1 as output with internal pullup
  TWI_MasterInit(&twiHmc, &TWIF, TWI_MASTER_INTLVL_MED_gc, TWI_BAUDSETTING);
  PORTF.PIN0CTRL = (PORTF.PIN0CTRL & ~PORT_OPC_gm) | PORT_OPC_WIREDANDPULL_gc;
  PORTF.PIN1CTRL = (PORTF.PIN1CTRL & ~PORT_OPC_gm) | PORT_OPC_WIREDANDPULL_gc;

  // HMC5883 and Data ready interrupt
    // Set up HMC by writing to conf. registers on chip
  hmc_SetRegister(&twiHmc, CONF_REG_A_adr, (measurement_normal_gc | rate_0_75_gc | average_8_gc));
  hmc_SetRegister(&twiHmc, CONF_REG_B_adr, gain_0_88_gc);

  // Enable data ready interrupt on PIN2.
  PORTF.PIN2CTRL = PORT_ISC_FALLING_gc;
  PORTF.INT0MASK = PIN2_bm;
  PORTF.INTCTRL = PORT_INT0LVL_LO_gc;
}

void init_itg3200() {
// 3-axis ITG3200 gyro sensor and TWI PORTE initialization

  //TWIE
    // Initiate TWI E, utilizing library.
    // Setting up E0 and E1 as output with internal pullup
  TWI_MasterInit(&twiItg, &TWIE, TWI_MASTER_INTLVL_MED_gc, TWI_BAUDSETTING);
  PORTE.PIN0CTRL = (PORTE.PIN0CTRL & ~PORT_OPC_gm) | PORT_OPC_WIREDANDPULL_gc;
  PORTE.PIN1CTRL = (PORTE.PIN1CTRL & ~PORT_OPC_gm) | PORT_OPC_WIREDANDPULL_gc;

  // ITG3200
    // Write configuration settings to registers
  itg_SetRegister(&twiItg, SMPLRT_DIV, 1); //Divider = (F_i/F_s)-1 = 1000/5-1
  itg_SetRegister(&twiItg, DLPF_FS, (Filter_42Hz | FullScale));
  itg_SetRegister(&twiItg, INT_CFG,
        ((IntConf_LogicLevel_bm & ActiveLevelHigh) |
         (IntConf_DriveType_bm & PushPull) |
         (IntConf_LatchMode_bm & LatchUntilIntIsCleard) |
         (IntConf_LatchClearMethod_bm & AnyRegisterRead) |
         (IntConf_EnableIntDeviceReady_bm & false) |
         (IntConf_EnableIntDataAvailable_bm & false)));
```

```
  itg_SetRegister(&twiItg, PWR_MGM, ClockSelect_PllWithXGyroReference);

 // Enable data ready interrupt on PIN2.
  PORTA.PIN2CTRL = PORT_ISC_RISING_gc;
  PORTA.INT0MASK = PIN2_bm;
  PORTA.INTCTRL  = PORT_INT0LVL_LO_gc;
}

void init_sar150() {
// 3x 1-axis gyro sensor initialization with separate SPI ports

  //Initiate SAR sensors on SPI ports
  sar_Init(&PORTD ,&spiSar[xAxis]); //X-axis PORT D
  sar_Init(&PORTF ,&spiSar[yAxis]); //Z-axis PORT F
  sar_Init(&PORTE ,&spiSar[zAxis]); //Y-axis PORT E

  //Disables SafeGuard on all three sensors.
  sar_SafeGuardDisable(&spiSar[xAxis], &spiDataPacketSar[xAxis]);
  sar_SafeGuardDisable(&spiSar[yAxis], &spiDataPacketSar[yAxis]);
  sar_SafeGuardDisable(&spiSar[zAxis], &spiDataPacketSar[zAxis]);
}

void init_SampleTimer(TC1_t * timer, uint16_t per) {
  // Sample clock initialization
  // Input: Clock to be set up
  //        Period in ticks (3.6 ticks is 1 ms)

  //Sample clock is set up with 3.6 ticks per ms.
  timer->PER = per-1; //Removing 1, since it starts on 0
  timer->CTRLA = TC_CLKSEL_DIV1024_gc;   //Clock is CPU/1024
  timer->CTRLB = TC_WGMODE_SS_gc;        //Single slope mode
  timer->INTCTRLA = TC_OVFINTLVL_LO_gc;  // Overflow interrupt
  timer->CTRLFSET = 0;          //set direction, 1-down 0-up
}

void init_RTC32() {
  // Real time clock initialization
  //
  // The real time clock is set up utilizing the internal
  // 1024 Hz divided signal of the internal 32.768 kHz RC osc.

  // Set periode to 1024*60sec=61440=1 min.
  do {
    RTC.PER = 61440; //One minute
  } while ((RTC.STATUS & RTC_SYNCBUSY_bm) == RTC_SYNCBUSY_bm);

  //1kHz internal RC oscillator.
  CLK.RTCCTRL = CLK_RTCSRC_RCOSC_gc | CLK_RTCEN_bm;

  //Prescale: 1. No effect.
  RTC.CTRL = RTC_PRESCALER_DIV1_gc;

  //Interrupt level HI
  RTC.INTCTRL = RTC_OVFINTLVL_HI_gc;

  // Sets time counters to 0
  realTimeClock.minute = 0;
  realTimeClock.hour = 0;
  realTimeClock.day = 0;
}

void init_coils() {
```

```
/****** About the PWM:
The PWM outputs consists of 3 pairs of outputs, one pair for each
axis. Each pair of outputs are connected to each separate
h-bridge ic. A pair is connected to the forward and reverse input
(FIN and BIN)on the h-bridge ic. Only one output in each pair is
active at a time, determed by the desired direction of the
magnet field created. The output wich is not active should be put
to low. The xAxis and yAxis outputs are controlled by the same
counter, TCD0. The zAxis outputs are controlled by the TCC1
*/

//Store pointer to the compare register into corresponding object
magnetorquer[xAxis].forwardOutput = &TCD0.CCA; //x Forward
magnetorquer[xAxis].reverseOutput = &TCD0.CCC; //x Reverse

magnetorquer[yAxis].forwardOutput = &TCD0.CCB; //y Forward
magnetorquer[yAxis].reverseOutput = &TCD0.CCD; //y Reverse

magnetorquer[zAxis].forwardOutput = &TCC1.CCB; //z Reverse
magnetorquer[zAxis].reverseOutput = &TCC1.CCA; //z Forward


// Configure Ports as output by setting corresponding bits in PORTx.DIRSET, also making shure output = 0
PORTD.DIRSET = PIN0_bm | PIN1_bm | PIN2_bm | PIN3_bm;  // x+y
PORTD.OUTCLR = PIN0_bm | PIN1_bm | PIN2_bm | PIN3_bm;

PORTC.DIRSET = PIN4_bm | PIN5_bm;            // # z
PORTC.OUTCLR = PIN4_bm | PIN5_bm;

// Setting the period wich should make an update rate of 20kHz-100kHz.
TCD0.PER = pwmPeriod;                  //#2
TCC1.PER = pwmPeriod;

//In CTRLB, the waveform is selected, and each compare channel is activated.
TCD0.CTRLB = TC_WGMODE_SS_gc | TC0_CCAEN_bm | TC0_CCBEN_bm | TC0_CCCEN_bm | TC0_CCDEN_bm;   // x+y
TCC1.CTRLB = TC_WGMODE_SS_gc | TC1_CCAEN_bm | TC1_CCBEN_bm;    // z

//Timer counter is started by setting a clocksource. System clock is choosed
TCD0.CTRLA = TC_CLKSEL_DIV1_gc;  //#5 x+y
TCC1.CTRLA = TC_CLKSEL_DIV1_gc;  //#5 z

//A start value of desiredValue is set.
magnetorquer[xAxis].desiredValue = 50;
magnetorquer[yAxis].desiredValue = 50;
magnetorquer[zAxis].desiredValue = 50;

//Make shure direction ports wich is not in use are disabled.
//THIS LINE CAN BE REMOVED WHEN NEXT HARDWARE VERSION DOES NOT CONNECT ANNYTHING TO PA3-PA5
//PORTA P3-P5 is connected to FIN on BD6210, but not in use!
PORTA.DIRCLR = PIN3_bm | PIN4_bm | PIN5_bm;
}

void init_adc(){
  /* About the ADC

  */

```

```
  //Make shure the portB is set to input..
  PORTB.DIRCLR = 0xFF;

  ADCB.CH0.MUXCTRL = (PIN0_bm<<3);
  ADCB.CH1.MUXCTRL = (PIN1_bm<<3);
  ADCB.CH2.MUXCTRL = (PIN2_bm<<3);


  ADCB.CH0.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;
  ADCB.CH1.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;
  ADCB.CH2.CTRL = ADC_CH_INPUTMODE_SINGLEENDED_gc;

  ADCB.CTRLA = ADC_ENABLE_bm;
  ADCB.CTRLB = ADC_RESOLUTION_12BIT_gc;
  ADCB.REFCTRL = ADC_REFSEL_INT1V_gc;
  ADCB.PRESCALER = ADC_PRESCALER_DIV4_gc;


}

// *******************  MAIN  **********************
int main (void) {
  // Main function initiates and stars the state machine loop

  // ***INITIALIZATION***

  //Sets my stream as stdout
  stdout = &mystdout;

  // Enable all interrupt levels.
  PMIC.CTRL |= PMIC_LOLVLEX_bm | PMIC_MEDLVLEX_bm | PMIC_HILVLEX_bm;

  // Enable global interrupts.
  sei();

  init_clk();      // System clock
  _delay_ms(50);
  init_RTC32();    // Real time clock
  init_uart();     // UART ports
  init_hmc5883(); // Magnetometer HMC5883
  init_itg3200(); // Gyro sensor ITG-3200
  init_sar150();  // Gyro sensors SAR150
  init_adc();      // Analog-Digital Converter
  init_coils();    // PWM output for coil control
  //Sample timer, def.: 500 ms*3.6=1800
  init_SampleTimer(SAMPLE_TIMER, 1800);

  // Prints a welcome message to nice human hyperterminal users
  printf("Welcome!\r\n"); //Sends start message to UART

  // Prints a status message to PC-client
  sendStatus();          //Sends status to UART

  // *** STATE MACHINE ***
  state = st_start;  //Initial state

  while(1) {
    //As long as a function is defined, loop
    for(uint8_t i=0; menu_state[i].pFunc ;i++) {
      //Find state array number of right state
      if (state == menu_state[i].state) {
        //Run function with corresponding array number.
```

```
        state = (menu_state[i].pFunc)();
        break;
      }
    }
  } //while end
}  //Main end!


// *** State Functions ***
/*
* All state functions:
*   -starts with "f_"
*   -have logic to determ next state (if needed)
*   -returns next state
*   -does not have any input variables
*/


state_t f_start(void) {
// Initial state, see flow chart

  // State machine logic
  if (sampleTimerCompareMatch) return st_sample;
  else if (commandReceived) return st_executeCommand;
  else if(1) return st_sleep;
  else return st_start;
}

state_t f_executeCommand(void) {
// This state is called when a terminate character is received on UART
// The received message is checked against command register

  // Available commands are stored here. The order of them
  // are important for the case structure.
  static char * commands[] =
   {"reset", "magstart", "magstop", "sarstart", "sarstop",
    "itgstart", "itgstop", "idle", "sample", "bdot", "extcont",
    "rate", "coilx", "coily", "coilz", "status"};
  bool valueReceived = false; // Is number value received?

  uint8_t i;
  uint16_t cmdValue;  // Number value are stored here if present.
  char receiveCmd[USART_RX_BUFFER_SIZE];   //Received COMMAND are put here
  char * receivePointer = receiveCmd;      //Pointer to receiveCmd
  char receiveValue[USART_RX_BUFFER_SIZE]; //Received VALUE are put here


  //Analyzing input string

  //1. Loop is saving one byte at a time to initially receiveCmd.
  //2. If "Line end" is received, array is ended and loop exits
  //3. If "space" is received, pointer is changed to value array
  //4. Exits with receiveCmd and if present receivedValue filled up
  for (i = 0; (i < USART_RX_BUFFER_SIZE) && (USART_RXBufferData_Available(&USART_data)); i++) {
    *receivePointer = USART_RXBuffer_GetByte(&USART_data); //1
    if (*receivePointer == 0x0D) { //2. If line end
      *receivePointer = 0x00; //
      break;
    }
    else if (*receivePointer == 0x20){  //3. If space
```

```
        *receivePointer = 0x00;
        receivePointer = receiveValue;
        valueReceived = true;
      }
      else {
        receivePointer++;
      }
    } //4. for loop end. Normally exitted because of break in if (2).

    // Translate string value to uint16_t value
    if (valueReceived) {
      cmdValue = atoi(receiveValue);
    }
    else {
      cmdValue = 0;
    }

    // Find the number in commands[] that match received command.
    for (i = arraysize(commands); i > 0; i--) {
        if (strcmp(receiveCmd, commands[i-1]) == 0) {
          break;
        }
      }

    // Utilize the recent found number to take action
    switch (i) {
      case 0:  //no match
        printf("Syntax error\r\n");
        break;
      case 1: //reset
        CPU_CCP=CCP_IOREG_gc;
        RST.CTRL=RST_SWRST_bm;
        break;
      case 2: //magstart
        sampleHmc = true;
        break;
      case 3: //magstop
        sampleHmc = false;
        break;
      case 4: //sarstart
        sampleSar = true;
        break;
      case 5: //sarstop
        sampleSar = false;
        break;
      case 6: //itgstart
        sampleItg = true;
        break;
      case 7: //itgstop
        sampleItg = false;
        break;
      case 8: //idle
        mode = idle;
        break;
      case 9: //sample
        mode = sample;
        break;
      case 10: //bdot
        mode = bDot;
        break;
      case 11: //extcont
        mode = externalControl;
```

```
        break;
    case 12: //rate
      // Set sample rate if it is a legal value
      if ((cmdValue > 0) && (cmdValue < 1001)) {
        printf("Time between sample is: %d ms\r\n",cmdValue);
        TC_SetPeriod(SAMPLE_TIMER, (uint16_t)(3.6 * cmdValue));
        TC_Reset(SAMPLE_TIMER);
      }
      else {
        printf("Rate must be set between 0 and 1000\r\n");
      }
      break;
    case 13: //coilx
      if ((cmdValue >= 0) && (cmdValue <= (pwmPeriod*2))) {
        magnetorquer[xAxis].desiredValue = cmdValue - pwmPeriod;
        mode = externalControl;
      }
    break;
    case 14: //coily
      if ((cmdValue >= 0) && (cmdValue <= (pwmPeriod*2))) {
        magnetorquer[yAxis].desiredValue = cmdValue - pwmPeriod;
        mode = externalControl;
      }
    break;
    case 15: //coilz
      if ((cmdValue >= 0) && (cmdValue <= (pwmPeriod*2))) {
        magnetorquer[zAxis].desiredValue = cmdValue - pwmPeriod;
        mode = externalControl;
      }
    break;
    case 16: //status
      sendStatus();
    break;
    default:
      break;
  }
  commandReceived = false;
  return st_start;
}

state_t f_sleep(void){

return st_start;
}

state_t f_sample(void) {
  // Performing sample on the sensors wich flag is enabled.
  // Is also doing preprocessing of aquired data.

  // Clear flag
  sampleTimerCompareMatch = false;

  //SAMPLE
  // SAR150 sample:
  if (sampleSar) {
    sar_DoThreeAxisMeasurement(spiSar, spiDataPacketSar, sarMeasurement);
    sarMeasurement[xAxis].rate.i16 =
    sarMeasurement[yAxis].rate.i16 * sarCalibtationParameter.misalignment↩
        [xAxis][yAxis] +
    sarMeasurement[zAxis].rate.i16 * sarCalibtationParameter.misalignment↩
        [xAxis][zAxis] +
```

```
    sarMeasurement[xAxis].rate.i16 * sarCalibtationParameter.scale[xAxis] +
    sarCalibtationParameter.bias[xAxis];

    sarMeasurement[yAxis].rate.i16 =
    sarMeasurement[xAxis].rate.i16 * sarCalibtationParameter.misalignment[yAxis][xAxis] +
    sarMeasurement[zAxis].rate.i16 * sarCalibtationParameter.misalignment[yAxis][zAxis] +
    sarMeasurement[yAxis].rate.i16 * sarCalibtationParameter.scale[yAxis] +
    sarCalibtationParameter.bias[yAxis];

    sarMeasurement[zAxis].rate.i16 =
    sarMeasurement[xAxis].rate.i16 * sarCalibtationParameter.misalignment[zAxis][xAxis] +
    sarMeasurement[yAxis].rate.i16 * sarCalibtationParameter.misalignment[zAxis][yAxis] +
    sarMeasurement[zAxis].rate.i16 * sarCalibtationParameter.scale[zAxis] +
    sarCalibtationParameter.bias[zAxis];
  }

  // ITG3200 sample:
  if (sampleItg) {
    itg_ReadSingleMeasurement(&twiItg, &itgMeasurement);
    itgMeasurement.rate[xAxis].i16 =
    itgMeasurement.rate[yAxis].i16 * itgCalibtationParameter.misalignment[xAxis][yAxis] +
    itgMeasurement.rate[zAxis].i16 * itgCalibtationParameter.misalignment[xAxis][zAxis] +
    itgMeasurement.rate[xAxis].i16 * itgCalibtationParameter.scale[xAxis] +
    itgCalibtationParameter.bias[xAxis];

    itgMeasurement.rate[yAxis].i16 =
    itgMeasurement.rate[xAxis].i16 * itgCalibtationParameter.misalignment[yAxis][xAxis] +
    itgMeasurement.rate[zAxis].i16 * itgCalibtationParameter.misalignment[yAxis][zAxis] +
    itgMeasurement.rate[yAxis].i16 * itgCalibtationParameter.scale[yAxis] +
    itgCalibtationParameter.bias[yAxis];

    itgMeasurement.rate[zAxis].i16 =
    itgMeasurement.rate[xAxis].i16 * itgCalibtationParameter.misalignment[zAxis][xAxis] +
    itgMeasurement.rate[yAxis].i16 * itgCalibtationParameter.misalignment[zAxis][yAxis] +
    itgMeasurement.rate[zAxis].i16 * itgCalibtationParameter.scale[zAxis] +
    itgCalibtationParameter.bias[zAxis];
  }

  // HMC5883 sample:
  if (sampleHmc) {
    hmc_SetRegister(&twiHmc, MODE_REG_adr, mode_single_measurement);
  }

  // State machine logic
  if (mode == bDot) return st_bDot;
  else if (mode == externalControl) return st_externalControl;
```

```
  else if (mode == sample) return st_print;

  else return st_start;
}

state_t f_externalControl(void) {
// This state does nothing, but is made as a suggestion for
// furthure implementation to other sub-systems.
// Functions performed in this state could be:
// -Send sample values to external controller
// -Receive coil control values from external controller

return st_activateMagnetorquer;
}

state_t f_print(void) {
// Prints sample values to UART. This is not performed in bDot mode

  static uint16_t sampleRealTime;

  /* Gets real time clock.
   * This functionality may be subject to change.
   * It may be considered to store a sample time
   * in data ready interrupt.
   * As of now, no requirements for the clock exists.
   */
  sampleRealTime = RTC.CNT;

  // Prints SAR150 sensor measurements
  if (sampleSar) {
    printf("SAR:%04x%04x%04x%04x%04x%04x%04x%04x\r\n",
           sarMeasurement[xAxis].rate.i16,
           sarMeasurement[yAxis].rate.i16,
           sarMeasurement[zAxis].rate.i16,
           sarMeasurement[xAxis].Temperature,
           realTimeClock.minute,
           sampleRealTime,
           sarMeasurement[yAxis].Temperature,
           sarMeasurement[zAxis].Temperature);

  }

  // Prints ITG-3200 sensor measurements
  if (sampleItg) {
    printf("ITG:%04x%04x%04x%04x%04x%04x\r\n",
           itgMeasurement.rate[xAxis].i16,
           itgMeasurement.rate[yAxis].i16,
           itgMeasurement.rate[zAxis].i16,
           itgMeasurement.temperature.i16,
           realTimeClock.minute,
           sampleRealTime);
  }

  // Prints HMC5883L sensor measurements
  if (sampleHmc) {
    printf("HMC:%04x%04x%04x0000%04x%04x\r\n",
           hmcMeasurement[xAxis].i16,
           hmcMeasurement[yAxis].i16,
           hmcMeasurement[zAxis].i16,
           realTimeClock.minute,
           sampleRealTime);
  }
```

```

return st_start;
}

state_t f_bDot(void) {
// B-Dot calculates control signal for the coils

  magnetorquer[xAxis].desiredValue = (int16_t)((bDotFactor/coilCurrent) * crossProduct1((int16_t*)&hmcMeasurement, (int16_t*)&itgMeasurement.rate));
  magnetorquer[yAxis].desiredValue = (int16_t)((bDotFactor/coilCurrent) * crossProduct2((int16_t*)&hmcMeasurement, (int16_t*)&itgMeasurement.rate));
  magnetorquer[zAxis].desiredValue = (int16_t)((bDotFactor/coilCurrent) * crossProduct3((int16_t*)&hmcMeasurement, (int16_t*)&itgMeasurement.rate));

  // printf("x:%d y:%d z:%d\r\n",magnetorquer[xAxis].desiredValue,magnetorquer[yAxis].desiredValue,magnetorquer[zAxis].desiredValue);

  return st_activateMagnetorquer;
}

state_t f_activateMagnetorquer(void) {
  /*
  1. Set wanted value
  2. read current
  3. calculate temperature compensation
  4. correct
  5. read new current value, store factor

  data contains
  magnetorquer[3]
  */


  for (uint8_t i = 0; i < 3; i++) { // Updating PWM on all three axis
    if (magnetorquer[i].desiredValue >= 0) { //FORWARD
      if (magnetorquer[i].desiredValue > pwmPeriod) {
        magnetorquer[i].desiredValue = pwmPeriod;
      }
      *magnetorquer[i].forwardOutput = (uint16_t)magnetorquer[i].desiredValue;
      *magnetorquer[i].reverseOutput = 0;
    }

    else {                                   //BACKWARD
      if (magnetorquer[i].desiredValue < -pwmPeriod) {
        magnetorquer[i].desiredValue = -pwmPeriod;
      }
      *magnetorquer[i].forwardOutput = 0;
      *magnetorquer[i].reverseOutput = (uint16_t)-magnetorquer[i].desiredValue;
    }
  }

  return st_start;
}


/*****************   INTERRUPT HANDLERS   ******************/

```

```

ISR(USARTC0_RXC_vect) {
/* UART Receive complete interrupt
 *
 * Calls the receive complete handler from USART library.
 * Sending pointer to correct USART as argument
 */

  USART_RXComplete(&USART_data);

  //Adding echo to the UART. Also adding carriage return to linefeed
  uint8_t temp = USART_data.buffer.RX[(USART_data.buffer.RX_Head-1) & ↩
      USART_TX_BUFFER_MASK];
  USART_TXBuffer_PutByte(&USART_data, temp);

  if (temp == 0x0D) {  //0x0D = Carriage return (\r)
    USART_TXBuffer_PutByte(&USART_data, 0x0A);  //0x0A = Line feed (\n)
    commandReceived = true;
  }
}

ISR(USARTC0_DRE_vect) {
/* Data register empty  interrupt
 * Calls the data register empty complete handler from USART library.
 * Sending pointer to correct USART as argument.
 */

  USART_DataRegEmpty(&USART_data);
}

ISR(TWIF_TWIM_vect) {
/* TWIF HMC5883 Master interrupt
 *
 * Calls the master interrupt handler from TWI library
 * Sending pointer to sensor twi as argument
 */

  TWI_MasterInterruptHandler(&twiHmc);
}

ISR(TWIE_TWIM_vect) {
/* TWIE ITG-3200 Master interrupt
 *
 * Calls the master interrupt handler from TWI library
 * Sending pointer to sensor twi as argument
 */

  TWI_MasterInterruptHandler(&twiItg);
}

ISR(PORTF_INT0_vect) {
/* Data Ready HMC5883 interrupt
 *
 * INT0 PORTF interrupt when data is ready on sensor
 * Calls Read measurement function from sensor library
 */

  hmc_ReadSingleMeasurement(&twiHmc, hmcMeasurement);
}

ISR(PORTA_INT0_vect) {
/* Data Ready ITG-3200 interrupt
```

```
 *
 * INT0 PORTA interrupt when data is ready on sensor
 * Calls Read measurement function from sensor library
 */

  itg_ReadSingleMeasurement(&twiItg, &itgMeasurement);
}

ISR(SPID_INT_vect) {
/* SPID SAR150 X-axis Master interrupt
 *
 * Calls the master interrupt handler from SPI library
 * Sending pointer to sensor SPI as argument
 */

  SPI_MasterInterruptHandler(&spiSar[xAxis]);
}

ISR(SPIF_INT_vect) {
/* SPIF SAR150 Y-axis Master interrupt
 *
 * Calls the master interrupt handler from SPI library
 * Sending pointer to sensor SPI as argument
 */

  SPI_MasterInterruptHandler(&spiSar[yAxis]);
}

ISR(SPIE_INT_vect) {
/* SPIE SAR150 Z-axis Master interrupt
 *
 * Calls the master interrupt handler from SPI library
 * Sending pointer to sensor SPI as argument
 */

  SPI_MasterInterruptHandler(&spiSar[zAxis]);
}

ISR(TCE1_OVF_vect) {
/* Sample clock interrupt
 *
 * Set sample flag. State machine will find out.
 */
  sampleTimerCompareMatch = true;
}

ISR(RTC_OVF_vect) {
/* Real time clock minute interrupt
 *
 * Run once every minute.
 * Counts up minutes, hours and days.
 * Unfortunately there is currently no way to set this watch.
 * Should be implemented with ODBC and other subsystems.
 */

  if(++realTimeClock.minute == 60) {
    realTimeClock.minute = 0;
    if (++realTimeClock.hour == 24) {
      realTimeClock.hour = 0;
      if (++realTimeClock.day == 366) {
        realTimeClock.day = 1;
      }
```

```
    }
  }
}

ISR(TCC1_CCA_vect) { //coil z Forward
  ADCB.CTRLA |= ADC_CH2START_bm;
  TCC1.INTCTRLB = TC_CCAINTLVL_OFF_gc | TC_CCBINTLVL_OFF_gc;
}

ISR(TCC1_CCB_vect) { //coil z Reverse
  ADCB.CTRLA |= ADC_CH2START_bm;
  TCC1.INTCTRLB = TC_CCAINTLVL_OFF_gc | TC_CCBINTLVL_OFF_gc;
}
ISR(TCD0_CCA_vect) { //coil x Forward
  ADCB.CTRLA |= ADC_CH0START_bm;
  TCD0.INTCTRLB = TC_CCAINTLVL_OFF_gc | TC_CCCINTLVL_OFF_gc;
}
ISR(TCD0_CCB_vect) { //coil y Forward
  ADCB.CTRLA |= ADC_CH1START_bm;
  TCD0.INTCTRLB = TC_CCBINTLVL_OFF_gc | TC_CCDINTLVL_OFF_gc;
}
ISR(TCD0_CCC_vect) { //coil x Reverse
  ADCB.CTRLA |= ADC_CH0START_bm;
  TCD0.INTCTRLB = TC_CCAINTLVL_OFF_gc | TC_CCCINTLVL_OFF_gc;
}

ISR(TCD0_CCD_vect) { //coil y Reverse
  ADCB.CTRLA |= ADC_CH1START_bm;
  TCD0.INTCTRLB = TC_CCBINTLVL_OFF_gc | TC_CCDINTLVL_OFF_gc;
}
```

Listing D.3: sar150.h

```
#include <avr/io.h>
#include "spi_driver.h"

//SPI-commands SAR150
#define  RARH    0b10000000
#define  RARLX   0b10001110
#define  RTMP    0b10110000
#define  RSR     0b10110100
#define  SGDIS1  0b01001110
#define  SGDIS2  0b01100011
#define  SGDIS3  0b00010010
#define  SGEN    0b01010101
#define  PRCEN   0b10101010

#define  SGDIS1_adr 0b11010111
#define  SGDIS2_adr 0b01010000
#define  SGDIS3_adr 0b10101000

#define xAxis 0
#define yAxis 1
#define zAxis 2
#define threeAxis 3

#define hasAddressByte true
#define noAddressByte false

typedef union sar_rate
  {
```

```c
    struct
      {
          uint8_t lsb;
          uint8_t msb;
      } b2;
    int16_t i16;
  } sar_rate_t;

typedef union sar_StatusRegister
  {
    struct
      {
        bool UNUSED :1;
        bool EXC_OK :1;
        bool DET_OK :1;
        bool PRNG_OK :1;
        bool ATEST_INACTIVE :1;
        bool OTPPAR_OK :1;
        bool SIG_OK :1;
        bool ADC_OK :1;
      } bools;
    uint8_t byte;
  } sar_StatusRegister_t;

typedef struct sarMeasurement
  {
    sar_rate_t rate;
    uint8_t Temperature;
    sar_StatusRegister_t Status;
  } sarMeasurement_t;

bool sar_DoThreeAxisMeasurement(SPI_Master_t * SPI_master, SPI_DataPacket_t * dataPacket, sarMeasurement_t * measurement);
void sar_ReadRegister(SPI_Master_t * SPI_master, SPI_DataPacket_t * dataPacket, sarMeasurement_t * measurement);
void sar_Init(PORT_t * port, SPI_Master_t * SPI_master);
void sar_SafeGuardDisable(SPI_Master_t * SPI_master, SPI_DataPacket_t * dataPacket);
void sar_SafeGuardEnable(SPI_Master_t * SPI_master, SPI_DataPacket_t * dataPacket);
```

Listing D.4: sar150c

```c
#include <avr/io.h>

#include "SAR150.h"
#include "spi_driver.h"
bool sar_DoThreeAxisMeasurement(SPI_Master_t * SPI_master, SPI_DataPacket_t * dataPacket, sarMeasurement_t * measurement) {
  sar_ReadRegister(&SPI_master[xAxis], &dataPacket[xAxis], &measurement[xAxis]);
  sar_ReadRegister(&SPI_master[yAxis], &dataPacket[yAxis], &measurement[yAxis]);
  sar_ReadRegister(&SPI_master[zAxis], &dataPacket[zAxis], &measurement[zAxis]);
  if (((measurement + xAxis)->Status.byte == 0xFF) &&
      ((measurement + xAxis)->Status.byte == 0xFF) &&
      ((measurement + xAxis)->Status.byte == 0xFF)) {
    return true;
  }
  else {
```

```
    return false;
  }
}

void sar_ReadRegister(SPI_Master_t * SPI_master, SPI_DataPacket_t * dataPacket, sarMeasurement_t * measurement) {
  const uint8_t sar_CmdReadString[] = {RARH, RARLX, RTMP, RSR, 0x00};
  uint8_t receivedData[5];

  SPI_MasterCreateDataPacket(dataPacket, noAddressByte, sar_CmdReadString,
                             receivedData, 5, SPI_master->port, PIN4_bm);

  /* Transmit and receive first data byte. */
  uint8_t status;
  do {
    status = SPI_MasterInterruptTransceivePacket(SPI_master, dataPacket);
  } while (status != SPI_OK);

  /* Wait for transmission to complete. */
  while (dataPacket->complete == false) {
  }

  if ((receivedData[1] == 0x80) && (receivedData[2] == 0x80)) { //ERROR
    measurement->rate.b2.msb = 0x5F;
  }
  else if ((receivedData[1] & 0x80) == 0x00) {  //Value is >0
    measurement->rate.b2.msb = (receivedData[1]>>4) & 0x0F;
  }
  else {  //Value is <0
    measurement->rate.b2.msb = (receivedData[1]>>4) | 0xF0;
  }
  measurement->rate.b2.lsb = (receivedData[1]<<4 & 0xF0) | (receivedData[2] & 0x0F);
  measurement->Temperature        = receivedData[3];
  measurement->Status.byte        = receivedData[4];
};

void sar_Init(PORT_t * port, SPI_Master_t * SPI_master) {
  SPI_t * SPI;
  if (port == &PORTC) SPI = &SPIC;
  else if (port == &PORTD) SPI = &SPID;
  else if (port == &PORTE) SPI = &SPIE;
  else SPI = &SPIF;

  port->DIRSET   = PIN4_bm;
  port->PIN4CTRL = PORT_OPC_TOTEM_gc;
  port->OUTSET   = PIN4_bm;

  SPI_MasterInit(SPI_master, SPI, port, false, SPI_MODE_0_gc,
                 SPI_INTLVL_MED_gc, false, SPI_PRESCALER_DIV4_gc);
};

void sar_SafeGuardDisable(SPI_Master_t * SPI_master, SPI_DataPacket_t * dataPacket) {
  uint8_t sar_ErrorHandling[] = {SGDIS1_adr, SGDIS1, SGDIS2_adr, SGDIS2, SGDIS3_adr, SGDIS3};
  uint8_t receivedData[6];

  SPI_MasterCreateDataPacket(dataPacket, hasAddressByte, sar_ErrorHandling,
                             receivedData, 6, SPI_master->port, PIN4_bm);
```

```

  /* Transmit and receive first data byte. */
  uint8_t status;
  do {
    status = SPI_MasterInterruptTransceivePacket(SPI_master, dataPacket);
  } while (status != SPI_OK);

  /* Wait for transmission to complete. */
  while (dataPacket->complete == false) {
  }
};

void sar_SafeGuardEnable(SPI_Master_t * SPI_master, SPI_DataPacket_t * ↩
    dataPacket) {
  uint8_t sar_ErrorHandling[] = {PRCEN};
  uint8_t receivedData[1];

  SPI_MasterCreateDataPacket(dataPacket, noAddressByte, sar_ErrorHandling↩
      ,
                             receivedData, 1, SPI_master->port, PIN4_bm);

  /* Transmit and receive first data byte. */
  uint8_t status;
  do {
    status = SPI_MasterInterruptTransceivePacket(SPI_master, dataPacket);
  } while (status != SPI_OK);

  /* Wait for transmission to complete. */
  while (dataPacket->complete == false) {
  }
};
```

Listing D.5: itg3200h

```
#include <avr/io.h>
#include "twi_master_driver.h"

#define Address_itg3200 0x69

#define xAxis 0
#define yAxis 1
#define zAxis 2
#define threeAxis 3


typedef union itg3200_16bitRegister
  {
    int16_t i16;
    struct
      {
          uint8_t lsb;
          uint8_t msb;
      } b2;
  } itg3200_16bitRegister_t;


typedef struct itgMeasurement
  {
      itg3200_16bitRegister_t  rate[3];
      itg3200_16bitRegister_t  temperature;
  } itgMeasurement_t;
```

```
typedef enum itg3200_adr_enum
{
    WHO_AM_I    = 0x00,
    SMPLRT_DIV  = 0x15,
    DLPF_FS     = 0x16,
    INT_CFG     = 0x17,
    INT_STATUS  = 0x1A,
    TEMP_OUT_H  = 0x1B,
    TEMP_OUT_L  = 0x1C,
    GYRO_XOUT_H = 0x1D,
    GYRO_XOUT_L = 0x1E,
    GYRO_YOUT_H = 0x1F,
    GYRO_YOUT_L = 0x20,
    GYRO_ZOUT_H = 0x21,
    GYRO_ZOUT_L = 0x22,
    PWR_MGM     = 0x3E
} itg3200_adr_enum_t;


// Defining DLPF_FS configuration register
#define DlpfFs_digitalLowPassFilter_gm  0x07
#define Filter_256Hz  (0x00<<0)
#define Filter_188Hz  (0x01<<0)
#define Filter_98Hz   (0x02<<0)
#define Filter_42Hz   (0x03<<0)
#define Filter_20Hz   (0x04<<0)
#define Filter_10Hz   (0x05<<0)
#define Filter_5Hz    (0x06<<0)

#define DlpfFs_fullScaleSelection_gm  0x18
#define FullScale     (0x03<<3)


// Defining interrupt configuration register
#define IntConf_LogicLevel_bm               0x80
#define IntConf_LogicLevel_bp               0x08
#define ActiveLevelLow                      0xFF
#define ActiveLevelHigh                     0x00

#define IntConf_DriveType_bm                0x40
#define IntConf_DriveType_bp                0x07
#define OpenDrain                           0xFF
#define PushPull                            0x00

#define IntConf_LatchMode_bm                0x20
#define IntConf_LatchMode_bp                0x06
#define LatchUntilIntIsCleard               0xFF
#define Pulse50us                           0x00

#define IntConf_LatchClearMethod_bm         0x10
#define IntConf_LatchClearMethod_bp         0x05
#define AnyRegisterRead                     0xFF
#define StatusRegisterReadOnly              0x00

#define IntConf_EnableIntDeviceReady_bm     0x04

#define IntConf_EnableIntDataAvailable_bm   0x01
```

```
// Defining interrupt status register
#define IntStatus_PllReady          0x01
#define IntStatus_RawDataIsReady    0x04

// Defining Power management register
#define PwrMgm_Reset_bm         0x80
#define PwrMgm_Reset_bp         8

#define PwrMgm_Sleep_bm         0x60
#define PwrMgm_Sleep_bp         7

#define PwrMgm_GyroXStandby_bm  0x40
#define PwrMgm_GyroXStandby_bp  6

#define PwrMgm_GyroYStandby_bm  0x20
#define PwrMgm_GyroYStandby_bp  5

#define PwrMgm_GyroZStandby_bm  0x08
#define PwrMgm_GyroZStandby_bp  4

#define ClockSelect_InternalOscillator     (0x00<<0)
#define ClockSelect_PllWithXGyroReference  (0x01<<0)
#define ClockSelect_PllWithYGyroReference  (0x02<<0)
#define ClockSelect_PllWithZGyroReference  (0x03<<0)
#define ClockSelect_PllWithExternal32k768  (0x04<<0)
#define ClockSelect_PllWithExternal19M2    (0x05<<0)

uint8_t itg_ReadRegister(TWI_Master_t *twi, itg3200_adr_enum_t registerAdr);
void itg_ReadSingleMeasurement(TWI_Master_t *twi, itgMeasurement_t *measurementData);
void itg_SetRegister(TWI_Master_t *twi, itg3200_adr_enum_t register_adr, uint8_t value);
```

Listing D.6: itg3200c

```
#include <avr/io.h>
#include "twi_master_driver.h"
#include "itg3200.h"


uint8_t itg_ReadRegister(TWI_Master_t *twi, itg3200_adr_enum_t registerAdr) {

  uint8_t SendBuffer = registerAdr;
  TWI_MasterWriteRead(twi, Address_itg3200, &SendBuffer, 1, 1);
  while (twi->status != TWIM_STATUS_READY) {
      // Wait until transaction is complete.
  }
  return twi->readData[0];
}


void itg_ReadSingleMeasurement(TWI_Master_t *twi, itgMeasurement_t *measurementData) {

  uint8_t sendBuffer = TEMP_OUT_H;
  TWI_MasterWriteRead(twi, Address_itg3200, &sendBuffer, 1, 8);
  while (twi->status != TWIM_STATUS_READY) {
      // Wait until transaction is complete.
  }
```

```
  measurementData->temperature.b2.msb = twi->readData[0];
  measurementData->temperature.b2.lsb = twi->readData[1];
  //Datasheet: tmp=-13200 at 35*C, Sensitivity 280 LSB/*C
  //Offset calculation: -13200-(35*280)=23000
  //New value scale factor = 280
  measurementData->temperature.i16 += 23000; //Fix offset
  measurementData->rate[xAxis].b2.msb = twi->readData[2];
  measurementData->rate[xAxis].b2.lsb = twi->readData[3];
  measurementData->rate[yAxis].b2.msb = twi->readData[4];
  measurementData->rate[yAxis].b2.lsb = twi->readData[5];
  measurementData->rate[zAxis].b2.msb = twi->readData[6];
  measurementData->rate[zAxis].b2.lsb = twi->readData[7];

}


void itg_SetRegister(TWI_Master_t *twi, itg3200_adr_enum_t register_adr, ↩
    uint8_t value) {

  uint8_t sendBuffer[] = {register_adr, value};
  TWI_MasterWriteRead(twi, Address_itg3200, sendBuffer, 2, 0);
  while (twi->status != TWIM_STATUS_READY) {
      // Wait until transaction is complete.
  }
}
```

Listing D.7: hmc5883.h

```
#include <avr/io.h>
#include "twi_master_driver.h"

#define ADDRESS_5883 0x1E

#define xAxis 0
#define yAxis 1
#define zAxis 2
#define threeAxis 3

typedef enum HMC5883_adr_enum
{
    CONF_REG_A_adr,
    CONF_REG_B_adr,
    MODE_REG_adr,
    DATA_OUT_X_MSB_REG_adr,
    DATA_OUT_X_LSB_REG_adr,
    DATA_OUT_Z_MSB_REG_adr,
    DATA_OUT_Z_LSB_REG_adr,
    DATA_OUT_Y_MSB_REG_adr,
    DATA_OUT_Y_LSB_REG_adr,
    STATUS_REG_adr,
    ID_REG_A_adr,
    ID_REG_B_adr,
    ID_REG_C_adr
} HMC5883_adr_enum_t;


/*  Configuration Register A start*/
  typedef enum hmc5883_measurement_mode
    {
        measurement_normal_gc = (0x00 << 0),
        measurement_positive_bias_gc = (0x01 << 0),
```

```c
        measurement_negative_bias_gc = (0x02 << 0)
    } hmc5883_measurement_mode_t;

  typedef enum hmc5883_typical_data_output_rate
    {
        rate_0_75_gc = (0x00 << 2),
        rate_1_5_gc = (0x01 << 2),
        rate_3_gc = (0x02 << 2),
        rate_7_5_gc = (0x03 << 2),
        rate_15_gc = (0x04 << 2),
        rate_30_gc = (0x05 << 2),
        rate_75_gc = (0x06  << 2)
    } hmc5883_typical_data_output_rate_t;

  typedef enum hmc5883_number_of_averaged_samples
    {
        average_1_gc = (0x00 << 5),
        average_2_gc = (0x01 << 5),
        average_4_gc = (0x02 << 5),
        average_8_gc = (0x03 << 5)
    } hmc5883_number_of_averaged_samples_t;
/*  Configuration Register A stop*/

/*  Configuration Register B start*/
  typedef enum hmc5883_gain
    {
        gain_0_88_gc = (0x00 << 5),
        gain_1_3_gc = (0x01 << 5),
        gain_1_9_gc = (0x02 << 5),
        gain_2_5_gc = (0x03 << 5),
        gain_4_0_gc = (0x04 << 5),
        gain_4_7_gc = (0x05 << 5),
        gain_5_6_gc = (0x06 << 5),
        gain_8_1_gc = (0x07 << 5)
    } hmc5883_gain_t;
/*  Configuration Register B stop*/

/*  Mode register start*/
  typedef enum hmc5883_operating_mode
    {
        mode_continuous_measurement = (0x00 << 0),
        mode_single_measurement = (0x01 << 0),
        mode_idle = (0x02 << 0)
    } hmc5883_operating_mode_t;
/*  Mode register stop*/



typedef union hmc_Measurement
  {
    int16_t i16;
    struct
    {
        uint8_t lsb;
        uint8_t msb;
    } b2;
  } hmc_Measurement_t;


uint8_t hmc_ReadRegister(TWI_Master_t *twi,
                              HMC5883_adr_enum_t register_adr);

```

```
void hmc_ReadSingleMeasurement(TWI_Master_t *twi,
                               hmc_Measurement_t *measurement_data);


void hmc_SetRegister(TWI_Master_t *twi,
                     HMC5883_adr_enum_t register_adr,
                     uint8_t value);
```

Listing D.8: hmc5883c

```
#include "HMC5883.h"


uint8_t hmc_ReadRegister(TWI_Master_t *twi, HMC5883_adr_enum_t register_adr) {
  uint8_t sendBuffer = register_adr;
  TWI_MasterWriteRead(twi, ADDRESS_5883, &sendBuffer, 1, 1);
  while (twi->status != TWIM_STATUS_READY) {
      /* Wait until transaction is complete. */
  }
  return twi->readData[0];
}


void hmc_ReadSingleMeasurement(TWI_Master_t *twi, hmc_Measurement_t *measurement_data) {
  uint8_t sendBuffer = DATA_OUT_X_MSB_REG_adr;
  TWI_MasterWriteRead(twi, ADDRESS_5883, &sendBuffer, 1, 6);
  while (twi->status != TWIM_STATUS_READY) {
      /* Wait until transaction is complete. */
  }
    measurement_data[xAxis].b2.msb = twi->readData[0];
    measurement_data[xAxis].b2.lsb = twi->readData[1];
    measurement_data[yAxis].b2.msb = twi->readData[2];
    measurement_data[yAxis].b2.lsb = twi->readData[3];
    measurement_data[zAxis].b2.msb = twi->readData[4];
    measurement_data[zAxis].b2.lsb = twi->readData[5];
}


void hmc_SetRegister(TWI_Master_t *twi, HMC5883_adr_enum_t register_adr, uint8_t value) {
  uint8_t sendBuffer[] = {register_adr, value};
  TWI_MasterWriteRead(twi, ADDRESS_5883, sendBuffer, 2, 0);
  while (twi->status != TWIM_STATUS_READY) {
      /* Wait until transaction is complete. */
  }
}
```

D.2 Coil Winder Card

Listing D.9: main.c

```
/*********************************************************************
*
*  File:        main.c
*  Description: Measure turns, control servo and provides menu-system
*  Project:     CubeSTAR Coil Winder Card
*  Target       ATmega169 on AVR Butterfly
*  Author:      Kjetil Rensel
```

```
* Revised:     August 2011
*
**********************************************************************/

#define F_CPU 1E6  //8MHz internal clock/8

#include <avr/io.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <util/delay.h>
#include <math.h>
#include <avr/eeprom.h>

#include "main.h"
#include "button.h"
#include "menu.h"
#include "LCD_Driver.h"

#define servoStepSetPoint  10
#define servoMmStep        10 //hundreds of mm
#define servoMin           500
#define servoMax           2200
#define servoRange         (servoMax-servoMin)
#define motorTicksPerTurn  594

#define radius   2320 //hundreds of mm
#define offset   215  //hundreds of mm
#define servoMmMax (offset*2)
#define servoMmMin 0

//Values in hundreds of mm
//Wire between 0.01 and 5 mm
#define wireStep  1
#define wireMax   500
#define wireMin   1

//Coil between 1 and 60 mm
#define coilStep  10
#define coilMax   6000
#define coilMin   100


#define toServo(_value)  OCR1A = _value;
#define counter0OverflowStatus ((TIFR0 & (1 << TOV0)) == (1 << TOV0))
#define rightIsPushed ((PINE & PIN3_MASK) == 0x00)
#define leftIsPushed  ((PINE & PIN2_MASK) == 0x00)
#define mmToEncodedPosition(_value) (_value / settings.wireThickness) * motorTicksPerTurn;
#define encodedPositionToServo() OCR1A = (acos((double)encodedPosition * radianMovePerTick)* servoRange);
#define servoMmToTicks(_value) (uint16_t)((double)(acos((offset - (double)_value) / offset) / M_PI * servoRange + servoMin))
#define servoTicksToMm(_value)  (uint16_t)(offset-cos((((double)_value-servoMin)/servoRange)*M_PI)*offset)
#define encodedPositionToMm(_value) (_value / (motorTicksPerTurn /settings.wireThickness) + servoMmToTicks(settings.leftPoint))

uint8_t nextstate;
static char *statetext;
```

```
uint8_t (*pStateFunc)(uint8_t);
uint8_t state, nextstate;
uint8_t input;

const settings_t settingsDefault = {
  //leftPoint   rightPoint   wireThickness   coilWith
    50,         350,         15,             200};
settings_t settings;
settings_t EEMEM settingsEeprom;
char printbuffer[20];
int16_t encodedPosition;
uint16_t totalTurns = 0;
uint16_t turnEncoderCounter = 0;
float radianMovePerTick; // (movePerTick/servoRange)*pi
uint32_t encoderCounter = 0;
uint32_t encoderLastServoUpdate = 0;
uint16_t temp;
uint16_t position;

enum servoDirection {
  left,
  right
}servoDirection = right;

enum lastValidPulse {
  high,
  low
}lastValidPulse = low;

enum display {
  servo,
  turns
}display = servo;


int main(void)  {
  uint8_t i;

  eeprom_read_block (&settings, &settingsEeprom, settingsByteLength);
  if ((settings.rightPoint == 0x0000) || (settings.leftPoint == 0x0000) || (settings.wireThickness == 0x0000) || (settings.coilWith == 0x0000)) {
    settings = settingsDefault;
  }
  updateRadianMovePerTick();

  sei();
  LCD_Init();

  DDRB |= (1 << DDB5);

  //Init clock
  CLKPR = 0x80; //enable write to register
  CLKPR = 0x03; //sets prescaler to div8

  //Init pwm
  TCCR1A = 0x82; // Clear OC1A on Compare Match, set OC1A at BOTTOM (non-inverting mode), WGM#14
  TCCR1B = 0x19; // Fast PWM, No prescaler

  ICR1 = 20000; //Sets top to 20000 (20ms) makes pulse to be sendt 50 times/s
```

```
  toServo(1500);

  //Init buttons as input
  PORTB = 0xFF;
  PORTE = 0xFF;

  Button_Init();

  // Initial state variables
  state = nextstate = st_welcome;
  statetext = mt_welcome;
  pStateFunc = NULL;


  while(1) {
    if (statetext) //Print text
    {
      LCD_puts_f(statetext);
      statetext = NULL;
    }

    input = getkey(); // Read buttons

    if (pStateFunc)
      {
          // When in this state, we must call the state function
          nextstate = pStateFunc(input);
      }

    else if (input != KEY_NULL)
      {
        // Plain menu, clock the state machine
        nextstate = StateMachine(state, input);
      }

    if (nextstate != state)
    {
      state = nextstate;
      for (i=0; menu_state[i].state; i++)
      {
        if (menu_state[i].state == state)
        {
            statetext =  menu_state[i].pText;
            pStateFunc = menu_state[i].pFunc;
            break;
        }
      }
    }
  }

 return 0;
}


/*  Function name : StateMachine
*   Returns :       nextstate
*   Parameters :    state, stimuli
*   Purpose :       Shifts between the different states
*/
uint8_t StateMachine(uint8_t state, uint8_t stimuli) {
    uint8_t nextstate = state;    // Default stay in same state
    uint8_t i;
```

```

    for (i=0; menu_nextstate[i].state; i++)
    {
        if (menu_nextstate[i].state == state && menu_nextstate[i].input ↩
            == stimuli)
        {
            // This is the one!
            nextstate = menu_nextstate[i].nextstate;
            break;
        }
    }

    return nextstate;
}

/*
*  The next seven functions, is run when in its corresponding state
*
*/


uint8_t f_Freerun(uint8_t input) {
  static bool enter = 1;

  if(enter) { //Entering
    position = servoTicksToMm(OCR1A);
    LCD_puts_f(mt_doFreerun);
    enter = 0;
  }

  else if(input == KEY_LEFT) { //Exiting
    enter = 1;
    return st_freerun;
  }

  else {   //Everytime else
    if (input == KEY_RIGHT) {
      position = 0;
    }
    f_step(&position, servoMmMax, servoMmMin, servoMmStep, &input);
    toServo(servoMmToTicks(position));
  }

  return st_doFreerun;
}

uint8_t f_SetLeftPoint(uint8_t input) {
  static bool enter = 1;

  if(enter) {
    position = servoTicksToMm(settings.leftPoint);
    LCD_puts_f(mt_doSetLeftPoint);
    enter = 0;
  }

  else if(input == KEY_LEFT) { //Exiting
    settings.leftPoint = servoMmToTicks(position);
    eeprom_write_word(&settingsEeprom.leftPoint, settings.leftPoint);
    updateRadianMovePerTick();
    enter = 1;
    return st_setLeftPoint;
  }
```

```

  else {   //Everytime else
    if (input == KEY_RIGHT) {
      position = servoTicksToMm(settings.leftPoint);
    }
    f_step(&position, servoMmMax, servoMmMin, servoMmStep, &input);
    toServo(servoMmToTicks(position));
  }

  return st_doSetLeftPoint;
}

uint8_t f_SetRightPoint(uint8_t input) {
  static bool enter = 1;

  if(enter) {
    position = servoTicksToMm(settings.rightPoint);
    LCD_puts_f(mt_doSetRightPoint);
    enter = 0;
  }

  else if(input == KEY_LEFT) { //Exiting
    settings.rightPoint = servoMmToTicks(position);
    eeprom_write_word(&settingsEeprom.rightPoint, settings.rightPoint);
    updateRadianMovePerTick();
    enter = 1;
    return st_setRightPoint;
  }

  else {   //Everytime else
    if (input == KEY_RIGHT) {
      position = servoTicksToMm(settings.rightPoint);
    }
    f_step(&position, servoMmMax, servoMmMin, servoMmStep, &input);
    toServo(servoMmToTicks(position));
  }

  return st_doSetRightPoint;
}

uint8_t f_SetWireThickness(uint8_t input) {
  static bool enter = 1;

  if(enter) { //Entering
    LCD_puts_f(mt_doSetWireThickness);
    enter = 0;
  }


  else if(input == KEY_LEFT) { //Exiting
    eeprom_write_word(&settingsEeprom.rightPoint, settings.rightPoint);
    updateRadianMovePerTick();
    enter = 1;
    return st_setWireThickness;
  }

  else {   //Everytime else
    if (input == KEY_RIGHT) {
      position = 0;
    }
    f_step(&settings.wireThickness, wireMax, wireMin, wireStep, &input);
  }
```

```

  return st_doSetWireThickness;
}

uint8_t f_SetCoilWith(uint8_t input) {
  static bool enter = 1;

  if(enter) { //Entering
    LCD_puts_f(mt_doSetCoilWith);
    enter = 0;
  }

  else if(input == KEY_LEFT) { //Exiting
    eeprom_write_word(&settingsEeprom.coilWith, settings.coilWith);
    updateRadianMovePerTick();
    enter = 1;
    return st_setCoilWith;
  }

  else {   //Everytime else
    if (input == KEY_RIGHT) {
      position = 0;
    }
    f_step(&settings.coilWith, coilMax, coilMin, coilStep, &input);
  }

  return st_doSetCoilWith;
}

uint8_t f_Run(uint8_t input) {
    static bool enter = 1;
    static uint16_t encodedPositionMax = 0;



  if(enter) {
    LCD_puts_f(mt_doRun);
    _delay_ms(500);
    TCCR0A = 0x03; //Prescaler 64
    PCMSK0 = 0x10; //Pin Change Mask Register 1, PC10 enable
    encodedPosition = mmToEncodedPosition(position);
    enter = 0;
    encodedPositionMax = servoTicksToMm(servoMax) * (motorTicksPerTurn / settings.wireThickness);
  }

  else if(input == KEY_PUSH) {
    PCMSK0 = PINE_MASK;
    TCCR0A = 0x00;
    enter = 1;
    return st_run;
  }

  if (encoderLastServoUpdate == encoderCounter) {  //No motion since last time -> MANUAL DRIVE
    if(rightIsPushed) {
      encoderLastServoUpdate -= 100;
      servoDirection = right;
    }
    else if(leftIsPushed){
      encoderLastServoUpdate -= 100;
      servoDirection = left;
```

```
    }
  }


  // Updating servo position
  if (servoDirection == right) {
      encodedPosition += encoderCounter - encoderLastServoUpdate;

      if (leftIsPushed) {  //Update RIGH point
        servoDirection = left;
        settings.rightPoint = OCR1A;
      }
    }
  else {  //servoDirection == left
      encodedPosition -= encoderCounter - encoderLastServoUpdate;

      if (rightIsPushed){  //Update LEFT point
        servoDirection = right;
        settings.leftPoint = OCR1A;
      }
    }

  encoderLastServoUpdate = encoderCounter;

  if (encodedPosition < 0) encodedPosition = 0;
  else if (encodedPosition > encodedPositionMax) encodedPosition = encodedPositionMax;
  position = encodedPositionToMm(encodedPosition);

  toServo(servoMmToTicks(position))


    //Updating servo right and left point
  if (position > servoTicksToMm(settings.rightPoint)) {           //servo passed right point
    if (rightIsPushed == false) {  // right is not pushed
      servoDirection = left;
    }
    else {  //right point is being overrided
      settings.rightPoint = OCR1A;

    }
  }
  else if (position < servoTicksToMm(settings.leftPoint)) {       //servo passed left poin
    if (leftIsPushed == false) { // left is not pushed
      servoDirection = right;
    }
    else {  //left point is being overrided
      settings.leftPoint = OCR1A;
    }
  }

  ultoa(((uint32_t)position) * 1000 + (uint32_t)totalTurns, printbuffer, 10);
  LCD_puts(printbuffer);


  return st_doRun;
}

uint8_t f_CounterReset(uint8_t input) {
```

```
  static bool enter = 1;

 if(enter) {
    totalTurns = 0;
    LCD_puts_f(mt_doCounterReset);
    enter = 0;
  }

  else if(input == KEY_LEFT) {
    enter = 1;
    return st_counterReset;
  }

  return st_doCounterReset;
}

void toLCD(uint16_t *value) {
  itoa(*value, printbuffer, 10);
  LCD_puts(printbuffer);
}

void updateRadianMovePerTick(void) {
  radianMovePerTick = (((settings.rightPoint-settings.leftPoint)*settings.wireThickness*M_PI)/(settings.coilWith*motorTicksPerTurn*servoRange));
 }

uint8_t f_step(uint16_t *variableToBeStepped, uint16_t max, uint16_t min, uint16_t step, uint8_t *input) {
  if(*input == KEY_UP) {
    if ((*variableToBeStepped + step) < max) {
      *variableToBeStepped += step;
    }
    else{
      *variableToBeStepped = max;
    }
  }

  else if(*input == KEY_DOWN) {
    if (*variableToBeStepped > (min + step)) {
      *variableToBeStepped -= step;
    }
    else{
      *variableToBeStepped = min;
    }
  }

  else if(*input == KEY_RIGHT) {
  }

  else if(*input == KEY_PUSH) {
  }

  else {
    return 0; //Did nothing
  }
  toLCD(variableToBeStepped);
  return 1; //Did something

}

//Interrupt run once every pulse from the motor. The
```

```
ISR(PCINT0_vect) {
  if(PCMSK0 == 0x10) {
    if(lastValidPulse == low){    //HIGH
      for(uint8_t i = 50; i > 0 ; i--) {
      }
      if ((PINE & PIN4_MASK) == PIN4_MASK) {
        PINB = PIN2_MASK;
        lastValidPulse = high;
        encoderCounter++;
        turnEncoderCounter++;
        if(turnEncoderCounter > motorTicksPerTurn){
          turnEncoderCounter = turnEncoderCounter - motorTicksPerTurn;
          totalTurns++;
        }
      }
    }

    else {
      for(uint8_t i = 50; i > 0 ; i--) {
      }
      if ((PINE & PIN4_MASK) == 0x00) {
        lastValidPulse = low;
      }
    }
  }
  else {
    PinChangeInterrupt();
  }
}

ISR(PCINT1_vect) {
    PinChangeInterrupt();
}
```

Listing D.10: main.h

```
/*******************************************************************
*
*  File:         main.h
*  Description:
*  Project:      CubeSTAR Coil Winder Card
*  Target        ATmega169 on AVR Butterfly
*  Author:       Kjetil Rensel
*  Revised:      August 2011
*
*******************************************************************/

//Button definitions

#define KEY_NULL    0
#define KEY_PUSH    1
#define KEY_LEFT    2
#define KEY_RIGHT   3
#define KEY_UP      4
#define KEY_DOWN    5


#define PIN0_MASK (1 << 0)
#define PIN1_MASK (1 << 1)
#define PIN2_MASK (1 << 2)
#define PIN3_MASK (1 << 3)
```

```
#define PIN4_MASK (1 << 4)
#define PIN5_MASK (1 << 5)
#define PIN6_MASK (1 << 6)

#define settingsByteLength 8

typedef struct settings
  {
    uint16_t leftPoint;
    uint16_t rightPoint;
    uint16_t wireThickness;
    uint16_t coilWith;
  } settings_t;

uint8_t StateMachine(uint8_t state, uint8_t stimuli);
uint8_t f_Freerun(uint8_t input);
uint8_t f_SetLeftPoint(uint8_t input);
uint8_t f_SetRightPoint(uint8_t input);
uint8_t f_SetWireThickness(uint8_t input);
uint8_t f_SetCoilWith(uint8_t input);
uint8_t f_Run(uint8_t input);
uint8_t f_CounterReset(uint8_t input);
void toLCD(uint16_t *value);
void updateRadianMovePerTick(void);
uint8_t f_step(uint16_t *variable, uint16_t max, uint16_t min, uint16_t ↩
    step, uint8_t *input);
```

Listing D.11: menu.h

```
/*******************************************************************
*
*  File:        menu.h
*  Description: Defines the states, state functions and menu texts
*  Project:     CubeSTAR Coil Winder Card
*  Target       ATmega169 on AVR Butterfly
*  Author:      Kjetil Rensel
*  Revised:     August 2011
*
********************************************************************/

#include <avr/io.h>
#include <avr/pgmspace.h>

//Menu states
#define st_welcome                 1
#define st_freerun                 2
#define st_setLeftPoint            3
#define st_setRightPoint           4
#define st_setWireThickness        5
#define st_setCoilWith             6
#define st_run                     7
#define st_counterResetConfirm     8
#define st_counterReset            9

#define st_doFreerun             20
#define st_doSetLeftPoint        21
#define st_doSetRightPoint       22
#define st_doSetWireThickness    23
#define st_doSetCoilWith         24
#define st_doRun                 25
#define st_doCounterReset        26
```

```

typedef struct
{
    uint8_t state;
    uint8_t input;
    uint8_t nextstate;
} menu_nextstate_t;


typedef struct
{
    uint8_t state;
    char *pText;
    uint8_t (*pFunc)(uint8_t input);
} menu_state_t;

//Menu text
char PROGMEM mt_welcome[]              = "Cubestar Coil winder by kjetil";
char PROGMEM mt_freerun[]              = "MANUAL";
char PROGMEM mt_setLeftPoint[]         = "LEFT    ";
char PROGMEM mt_setRightPoint[]        = "RIGHT   ";
char PROGMEM mt_setWireThickness[]     = "WIRE    ";
char PROGMEM mt_setCoilWith[]          = "COIL    ";
char PROGMEM mt_run[]                  = "RUN";
char PROGMEM mt_counterReset[]         = "RESET COUNTER";
char PROGMEM mt_counterResetConfirm[]= "ARE YOU SHURE, LEFT TO CANCEL";
char PROGMEM mt_doFreerun[]            = "UP/DWN";
char PROGMEM mt_doSetLeftPoint[]       = "UP/DWN";
char PROGMEM mt_doSetRightPoint[]      = "UP/DWN";
char PROGMEM mt_doSetWireThickness[] = "UP/DWN";
char PROGMEM mt_doSetCoilWith[]        = "UP/DWN";
char PROGMEM mt_doRun[]                = "POSCNT";
char PROGMEM mt_doCounterDisplay[]     = "UP/DWN";
char PROGMEM mt_doCounterReset[]       = "CONTER RESETTED, LEFT TO GO BACK";


menu_nextstate_t menu_nextstate[] = {
//  STATE                       INPUT       NEXT STATE
    {st_welcome,                KEY_UP,     st_freerun},
    {st_welcome,                KEY_DOWN,   st_freerun},
    {st_welcome,                KEY_RIGHT,  st_freerun},
    {st_welcome,                KEY_LEFT,   st_freerun},

    {st_freerun,                KEY_UP,     st_counterReset},
    {st_freerun,                KEY_DOWN,   st_setLeftPoint},
    {st_freerun,                KEY_RIGHT,  st_doFreerun},
    {st_freerun,                KEY_LEFT,   st_freerun},

    {st_setLeftPoint,           KEY_UP,     st_freerun},
    {st_setLeftPoint,           KEY_DOWN,   st_setRightPoint},
    {st_setLeftPoint,           KEY_RIGHT,  st_doSetLeftPoint},
    {st_setLeftPoint,           KEY_LEFT,   st_setLeftPoint},

    {st_setRightPoint,          KEY_UP,     st_setLeftPoint},
    {st_setRightPoint,          KEY_DOWN,   st_setWireThickness},
    {st_setRightPoint,          KEY_RIGHT,  st_doSetRightPoint},
    {st_setRightPoint,          KEY_LEFT,   st_setRightPoint},

    {st_setWireThickness,       KEY_UP,     st_setRightPoint},
    {st_setWireThickness,       KEY_DOWN,   st_setCoilWith},
    {st_setWireThickness,       KEY_RIGHT,  st_doSetWireThickness},
```

```
    {st_setWireThickness ,      KEY_LEFT ,   st_setWireThickness},

    {st_setCoilWith ,           KEY_UP ,     st_setWireThickness},
    {st_setCoilWith ,           KEY_DOWN ,   st_run},
    {st_setCoilWith ,           KEY_RIGHT ,  st_doSetCoilWith},
    {st_setCoilWith ,           KEY_LEFT ,   st_setCoilWith},

    {st_run ,                   KEY_UP ,     st_setCoilWith},
    {st_run ,                   KEY_DOWN ,   st_counterReset},
    {st_run ,                   KEY_RIGHT ,  st_doRun},
    {st_run ,                   KEY_LEFT ,   st_run},


    {st_counterReset ,          KEY_UP ,     st_run},
    {st_counterReset ,          KEY_DOWN ,   st_freerun},
    {st_counterReset ,          KEY_RIGHT ,  st_counterResetConfirm},
    {st_counterReset ,          KEY_LEFT ,   st_counterReset},


    {st_counterResetConfirm ,   KEY_UP ,     st_counterResetConfirm},
    {st_counterResetConfirm ,   KEY_DOWN ,   st_counterResetConfirm},
    {st_counterResetConfirm ,   KEY_RIGHT ,  st_doCounterReset},
    {st_counterResetConfirm ,   KEY_LEFT ,   st_counterReset},


    {0 ,                        0 ,          0}
};

menu_state_t menu_state[] = {
//  STATE                      STATE TEXT                STATE_FUNC
    {st_welcome              , mt_welcome              , NULL},
    {st_freerun              , mt_freerun              , NULL},
    {st_setLeftPoint         , mt_setLeftPoint         , NULL},
    {st_setRightPoint        , mt_setRightPoint        , NULL},
    {st_setWireThickness     , mt_setWireThickness     , NULL},
    {st_setCoilWith          , mt_setCoilWith          , NULL},
    {st_run                  , mt_run                  , NULL},
    {st_counterResetConfirm  , mt_counterResetConfirm  , NULL},
    {st_counterReset         , mt_counterReset         , NULL},
    {st_doFreerun            , NULL                    , f_Freerun},
    {st_doSetLeftPoint       , NULL                    , f_SetLeftPoint},
    {st_doSetRightPoint      , NULL                    , f_SetRightPoint},
    {st_doSetWireThickness   , NULL                    , f_SetWireThickness↩
        },
    {st_doSetCoilWith        , NULL                    , f_SetCoilWith},
    {st_doRun                , NULL                    , f_Run},
    {st_doCounterReset       , NULL                    , f_CounterReset},

    {0                       , NULL                    , NULL}
};
```

Listing D.12: button.c

```
/*********************************************************************
*
*  File:         button.c
*  Description: Reads the button input
*  Project:      CubeSTAR Coil Winder Card
*  Target        ATmega169 on AVR Butterfly
*  Author:       ATMEL
*  Rewritten:    Kjetil Rensel
```

```
* Revised:    August 2011
*
*****************************************************************/

#include <avr/io.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>
#include <avr/interrupt.h>

#include "button.h"

uint8_t KEY;
bool KEY_VALID = false;



// Initializes the five button pin
void Button_Init(void)
{
  // Enable pin change interrupt on PORTB and PORTE
  PCMSK0 = PINE_MASK;
  PCMSK1 = PINB_MASK;
  EIFR = (1<<PCIF0)|(1<<PCIF1);
  EIMSK = (1<<PCIE0)|(1<<PCIE1);

}



//Check status on the joystick
void PinChangeInterrupt(void)
{
    uint8_t buttons;
    uint8_t key;

/*
    Read the buttons:

    Bit             7   6   5   4   3   2   1   0
    ---------------------------------------------
    PORTB           B   A       O
    PORTE                           D   C
    ---------------------------------------------
    PORTB | PORTE   B   A       O   D   C
    =============================================
*/


    buttons = (~PINB) & PINB_MASK;
    buttons |= (~PINE) & PINE_MASK;

    // Output virtual keys
    if (buttons & (1<<BUTTON_UP))
        key = KEY_UP;
    else if (buttons & (1<<BUTTON_DOWN))
        key = KEY_DOWN;
    else if (buttons & (1<<BUTTON_LEFT))
        key = KEY_LEFT;
    else if (buttons & (1<<BUTTON_RIGHT))
        key = KEY_RIGHT;
    else if (buttons & (1<<BUTTON_PUSH))
```

```
        key = KEY_PUSH;
    else
        key = KEY_NULL;


    if((key != KEY_NULL) && !KEY_VALID)
    {
      KEY = key;          // Store key in global key buffer
      KEY_VALID = true;
    }

    EIFR = (1<<PCIF1) | (1<<PCIF0);  // Delete pin change interrupt flags
}


//Get the valid key and returns it
uint8_t getkey(void)
{
    uint8_t k;

    cli();

    if (KEY_VALID)              // Check for unread key in buffer
    {
        k = KEY;
        KEY_VALID = false;
    }
    else
        k = KEY_NULL;           // No key stroke available

    sei();

    return k;
}
```

Listing D.13: button.h

```
/********************************************************************
*
*  File:        button.h
*  Description: Defines the propertis of buttons connected
*  Project:     CubeSTAR Coil Winder Card
*  Target       ATmega169 on AVR Butterfly
*  Author:      Kjetil Rensel
*  Revised:     August 2011
*
*******************************************************************/


#define PINB_MASK ((1<<PINB4)|(1<<PINB6)|(1<<PINB7))
#define PINE_MASK ((1<<PINE2)|(1<<PINE3))

#define BUTTON_UP      6   // UP
#define BUTTON_DOWN    7   // DOWN
#define BUTTON_LEFT    2   // LEFT
#define BUTTON_RIGHT   3   // RIGHT
#define BUTTON_PUSH    4   // PUSH

//Button definitions

#define KEY_NULL    0
```

```
#define KEY_PUSH    1
#define KEY_LEFT    2
#define KEY_RIGHT   3
#define KEY_UP      4
#define KEY_DOWN    5

void PinChangeInterrupt(void);
void Button_Init(void);
uint8_t getkey(void);
```

Appendix E

Matlab Source Code

E.1 Gyro Calibration

Listing E.1: Gyro_calibration.m Code file for the GUI gyro calibrating software

```
%Matlab Gyro_calibration code
%Written By: Kjetil Rensel
%2011

%Button: Reads the path and file name of X-axis spin file
function pushbutton1_Callback(hObject, ~, handles)
global fileNameX;
global pathX;
[fileNameX pathX] = uigetfile('*.txt');
set(handles.textMeasX, 'String', fileNameX);
guidata(hObject, handles);
%set(handles.editFileName1, 'string', fileName);

%Button: Reads the path and file name of Y-axis spin file
function pushbutton2_Callback(hObject, ~, handles)
global fileNameY;
global pathY;
[fileNameY pathY] = uigetfile('*.txt');
set(handles.textMeasY, 'String', fileNameY);
guidata(hObject, handles);

%Button: Reads the path and file name of Z-axis spin file
function pushbutton3_Callback(hObject, ~, handles)
global fileNameZ;
global pathZ;
[fileNameZ pathZ] = uigetfile('*.txt');
set(handles.textMeasZ, 'String', fileNameZ);
guidata(hObject, handles);

% Button: Latex plots, sets plot properties to latex design.
function pushbutton5_Callback(hObject, ~, handles)
set(0,'DefaultTextInterpreter','latex');
set(0,'DefaultTextFontSize',16);
set(0,'DefaultFigurePosition', [500 300 800 500]);
guidata(hObject, handles);


% Button: Default plots, sets plot properties to default.
```

```matlab
function pushbutton6_Callback(hObject, ~, handles)
set(0,'DefaultTextInterpreter','none');
set(0,'DefaultTextFontSize',12);
guidata(hObject, handles);

%Button: Open figure "Temperature" in separate window
function pushbutton9_Callback(~, ~, ~)
figure();
temperaturePlot();

%Button: Run Kalman filtering, and plots the error plot.
function pushbutton11_Callback(~, ~, handles)

global fileNameX;
global fileNameY;
global fileNameZ;
global pathX;
global pathY;
global pathZ;
global tempDep %%tempDep([x y z],[TemperatureCoefficient Offset])
global meas_cal_e;
global meas_e;
global s;
global x_store;

raw_x = importdata(strcat(pathX,fileNameX));
raw_y = importdata(strcat(pathY,fileNameY));
raw_z = importdata(strcat(pathZ,fileNameZ));

s(1) = size(raw_x,1);
s(2) = s(1) + size(raw_y,1);
s(3) = s(2) + size(raw_z,1);

%%%Convert to common format from chip specific
if (size(raw_x,2) == 7) %ITG
    %Format is: <X><Y><Z><tempX><reference>
meas = [raw_x(:,2:4), raw_x(:,5)
        raw_y(:,2:4), raw_y(:,5)
        raw_z(:,2:4), raw_z(:,5)];
meas = [meas(:,1:3)/14.375 meas(:,4)/280];%%ITG specific scale factor

ref=7;

else %SAR
meas = [raw_x(:,2:4), (raw_x(:,5)+raw_x(:,6)+raw_x(:,7))/3, raw_x(:,9)
        raw_y(:,2:4), (raw_y(:,5)+raw_y(:,6)+raw_y(:,7))/3, raw_y(:,9)
        raw_z(:,2:4), (raw_z(:,5)+raw_z(:,6)+raw_z(:,7))/3, raw_z(:,9)];
meas(:,1:3) = meas(:,1:3)/10;%%SAR specific scale factor
ref=9;
end

%Creating spintable reference variable
reference = zeros(s(3),3);
reference(1:s(1),1)       = raw_x(:,ref);
reference(s(1)+1:s(2),2) = raw_y(:,ref);
reference(s(2)+1:s(3),3) = raw_z(:,ref);

%%%Pre-Processing (temperature/offset compensating)
meas_pp=zeros(s(3),3);

for i=1:s(3)
```

```
    meas_pp(i,:) = (meas(i,1:3)' + (-tempDep(:,1) .* meas(i,4)) - tempDep(:,2))';
end

%%%Calculates error
meas_e = zeros(s(3),3);
meas_e = reference - meas_pp;

axes(handles.axesError);
errorPlot(0);

%%%%%%%%%%%%%%%%%%%%%%%%%KALMAN
Omega=zeros(3,9,s(3)); %Empty omega

%Fill up Omega with pre processed measurement
Omega(1,1,:)=meas_pp(:,2);
Omega(1,2,:)=meas_pp(:,3);
Omega(2,3,:)=meas_pp(:,1);
Omega(2,4,:)=meas_pp(:,3);
Omega(3,5,:)=meas_pp(:,1);
Omega(3,6,:)=meas_pp(:,2);
Omega(1,7,:)=meas_pp(:,1);
Omega(2,8,:)=meas_pp(:,2);
Omega(3,9,:)=meas_pp(:,3);

H=zeros(3,12,s(3));

for i=1:s(3)
    H(:,:,i)=[Omega(:,:,i) -eye(3)];
end

x=ones(12,1);
R=eye(3);
P=diag(x)^2;
x_store=zeros(s(3),12);

%The Kalman Loop starts here:
for i=1:s(3)
    x_store(i,:)=x;
    %%Time Update -PREDICT-
    %x=x and p=p

    %%Measurement Update -CORRECT-
    K = P * H(:,:,i)' / (H(:,:,i) * P * H(:,:,i)' + R);
    x = x + K * (meas_e(i,:)'-H(:,:,i) * x);
    P = P - K * H(:,:,i) * P;
end

%%Plot kalman:
axes(handles.axesKalman);
kalmanPlot();

%%%%%%%%%%%%%%%%%%%%%%%%%KALMAN END

%Write parameters to table
tablData=get(handles.uitableParameters,'Data');
tablData=[tablData(1:2,:); 0 x(1:2)'; x(3) 0 x(4); x(5:6)' 0; x(7:9)'; x(10:12)'];
set(handles.uitableParameters,'Data',tablData);

%%%Correcting data, and make a meas_cal variable
S = (eye(3) + diag(x(7:9)) + [0    x(1) x(2) ;...
```

```
                              x(3) 0    x(4) ;...
                              x(5) x(6) 0]   );
meas_cal = zeros (s(3),3);
for i=1:s(3)
    meas_cal(i,:) = (S*meas_pp(i,:)' - x(10:12))';
end

meas_cal_e = reference - meas_cal;

%Plot the error after correction
axes(handles.axesError);
errorPlot(1);

%%Wrinting variables to workspace
assignin('base', 'x_store', x_store);
assignin('base', 'tempDep', tempDep);
assignin('base', 'meas', meas);
assignin('base', 'meas_pp', meas_pp);
assignin('base', 'meas_e', meas_e);
assignin('base', 'meas_cal', meas_cal);
assignin('base', 'reference', reference);
assignin('base', 'x', x);

%Button: CHanges the Error plot window to display error before calibration
function pushbutton12_Callback(~, ~, handles)
axes(handles.axesError);
errorPlot(0);

%Button: CHanges the Error plot window to display error after calibration
function pushbutton13_Callback(~, ~, handles)
axes(handles.axesError);
errorPlot(1);


%Variable list: When a variable is choosen, the
%Temperature test data is plotted, and a linear fit is performed.
function popupmenu1_Callback(hObject, ~, handles)

update_popup(handles);
global tempMeas;
global tempDep; %%tempDep([x y z],[TemperatureCoefficient Offset])

vars = get(hObject,'String');
%tempMeas = evalin('base',strcat(vars(get(hObject,'Value'))));

tempMeas = evalin('base',vars{get(hObject,'Value')});
if(size(tempMeas,2)==6)
    tempDep(1,:) = polyfit(tempMeas(:,4), tempMeas(:,1),1);
    tempDep(2,:) = polyfit(tempMeas(:,5), tempMeas(:,2),1);
    tempDep(3,:) = polyfit(tempMeas(:,6), tempMeas(:,3),1);
elseif(size(tempMeas,2)==4)
    tempDep(1,:) = polyfit(tempMeas(:,4), tempMeas(:,1),1);
    tempDep(2,:) = polyfit(tempMeas(:,4), tempMeas(:,2),1);
    tempDep(3,:) = polyfit(tempMeas(:,4), tempMeas(:,3),1);
end

set(handles.uitableParameters,'Data',tempDep');

axes(handles.axesTmp);
cla;
temperaturePlot();
```

```
%Button: Performs a update of the variable list.
%This must be done when a new variable is made
%in the workspace after GUI is opened
function pushbutton17_Callback(~, ~, handles)
update_popup(handles)

function update_popup(handles)
vars = evalin('base','who');
set(handles.popupmenu1,'String',vars)



%Button: Open figure "Kalman" in separate window
function pushbutton19_Callback(~, ~, ~)
figure();
kalmanPlot();

%Button: Open figure "Error" in separate window
function pushbutton20_Callback(~, ~, ~)
global errorplotnr;
figure();
errorPlot(errorplotnr);

%Function: plots "temperature" figure
function temperaturePlot(~)
global tempMeas;
global tempDep;
title('SAR150-100 Temperature Dependent Bias','FontSize',14);
ylabel('Bias (deg/sec)','FontSize',14);
xlabel('Temperature (deg/C)','FontSize',14);
hold on;
if(size(tempMeas,2)==6) %%SAR
    scatter(tempMeas(:,4), tempMeas(:,1),'.');
    scatter(tempMeas(:,5), tempMeas(:,2),'.');
    scatter(tempMeas(:,6), tempMeas(:,3),'.');
elseif(size(tempMeas,2)==4) %%ITG
    scatter(tempMeas(:,4), tempMeas(:,1),'.');
    scatter(tempMeas(:,4), tempMeas(:,2),'.');
    scatter(tempMeas(:,4), tempMeas(:,3),'.');
end
set(legend('X-axis','Y-axis','Z-axis'),'Interpreter','latex','FontSize',14)
a=axis;
line([a(1) a(2)],[a(1)*tempDep(1,1)+tempDep(1,2) a(2)*tempDep(1,1)+tempDep(1,2)],'Color',[.5 .5 .5],'LineWidth',2);
line([a(1) a(2)],[a(1)*tempDep(2,1)+tempDep(2,2) a(2)*tempDep(2,1)+tempDep(2,2)],'Color',[.5 .5 .5],'LineWidth',2);
line([a(1) a(2)],[a(1)*tempDep(3,1)+tempDep(3,2) a(2)*tempDep(3,1)+tempDep(3,2)],'Color',[.5 .5 .5],'LineWidth',2);
axis(a); clear a;

%Function: plots "Error" figure
function errorPlot(calibrated)
global meas_cal_e;
global meas_e;
global s;
global errorplotnr;

errorplotnr = calibrated;

if calibrated == 1
```

```
    plot(meas_cal_e);
    title('Rate Table Test Errors After Calibration','FontSize',14);
else
    plot(meas_e);
    title('Rate Table Test Error After Preprocessing','FontSize',14);
end;

line([0 s(3)],[0 0], 'Color', 'k');
xlabel('Time (samples)','FontSize',14);
ylabel('Angular Velocity (deg/sec)','FontSize',14);
set(legend('X-axis','Y-axis','Z-axis'),'Interpreter','latex','FontSize',14)

%Function: Plots "Kalman" function
function kalmanPlot()
global x_store;
global s;
plot(x_store);
title('Kalman State Parameters','FontSize',14);
set(legend('$\delta_{xy}$', '$\delta_{xy}$', '$\delta_{yx}$',...
    '$\delta_{yz}$', '$\delta_{zx}$', '$\delta_{zy}$',...
    '$\lambda_{x}$', '$\lambda_{y}$', '$\lambda_{z}$',...
    '$\beta_{x}$', '$\beta_{y}$', '$\beta_{z}$'),'Interpreter',...
    'latex','FontSize',14);
xlabel('Iterations','FontSize',14);
ylabel('State Values','FontSize',14);
```

E.2 Magnetometer Calibration

Listing E.2: calibrate.m Runs the calibration process based on the measurement data

```
%%%%%%%%MagnetometerCalibration
% Runs the optimalization process as many times as defined below
% If executed again, it continues to minimize the function.
%
% Written by Kjetil Rensel
% Based on the paper "A Geometric Approach to Strapdown Magnetometer
% Calibration in Sensor Frame"
s = size(h,2);
it=1500; %NUMBER OF ITERATIONS  <---

mini = [ min(h(1,:)) min(h(2,:)) min(h(3,:)) ];%finding min on each axis
maxi = [ max(h(1,:)) max(h(2,:)) max(h(3,:)) ];%finding max on each axis

%%If b, T, g_b and g_T not exists, make a good guess:
if ((exist('b')+exist('T')+exist('g_b')+exist('g_T')) ~= 4)
    b = [(maxi(1)+mini(1))/2 (maxi(2)+mini(2))/2 (maxi(3)+mini(3))/2]';
    T = inv(diag([(maxi(1)-mini(1)) (maxi(2)-mini(2)) (maxi(3)-mini(3))]/2));
    g_b = 0.000001;
    g_T = 0.000001;
end

d_T = zeros(9,s);
d_b = zeros(3,s);
c_T = zeros(1,s);
d_T_sum = zeros(3,3,it);
d_b_sum = zeros(3,1,it);

err=zeros(1,it);
```

```
for i=1:it;

    u = h - repmat(b,1,s); %%Creating u with b and measurement

    TT=(T'*T);
    for j=1:s    %%Gradient of T loop
        c_T(j) = 1-1/norm(T*u(:,j));
        d_T(:,j) = (2 * c_T(j)) * ( kron(u(:,j), T*u(:,j)) );
        d_T_sum(:,:,i) = d_T_sum(:,:,i) + reshape(d_T(:,j),3,3);
    end
    T_old=T;
    T = T - d_T_sum(:,:,i) .* g_T; %Determ new T

    for j=1:s
        err(i) = err(i) + ( norm(T*(h(:,j)-b)) -1)^2; %Calculate error
    end


    if ((i > 1) && (err(i-1) < err(i)))%Decide whether new or old T is best
        T=T_old;
        g_T=g_T*0.9;
    else
        g_T=g_T*1.01;
    end

    for j=1:s    %%Gradient of b loop
        c_T(j) = 1-1/norm(T*u(:,j));
        d_b(:,j) = (-2 * c_T(j)) * TT * u(:,j);
        d_b_sum(:,:,i) = d_b_sum(:,:,i) + reshape(d_b(:,j),3,1);
    end

    b_old=b;
    b = b - d_b_sum(:,:,i) .* g_b;%Determ new B

    err(i) = 0;
    for j=1:s
        err(i) = err(i)+ ( norm(T*(h(:,j)-b)) -1)^2; %Calculate error
    end


    if ((i > 1) && (err(i-1) < err(i)))%Decide whether new or old b is best
         b=b_old;
         g_b=g_b*0.9;
    else
       g_b=g_b*1.01;
    end
end

[U,S,R]=svd(T); %SVD decomposition

S=(S/(norm([S(1,1) S(2,2) S(3,3)]))); %%Keep the absolute value
h_calib=zeros(3,size(h,2));
for i=1:size(h,2)
    h_calib(:,i)=S*R'*(h(:,i)-b);   %%Correct data!
end

set(figure(), 'Position', [500 300 800 500]);
plot(err);
```

```
title('Total error of the minimization function','interpreter','latex','fontsize',14)
xlabel('Iterations','interpreter','latex','fontsize',14)
ylabel('Sum of Errors','interpreter','latex','fontsize',14)
```

Listing E.3: plot_abs.m Plots absolute value of magnetometer measurement data

```
function mag_abs = plot_abs(meas)
mag_abs=zeros(size(meas,2),1);
for i=1:size(meas,2)
    mag_abs(i)=norm(meas(:,i));
end

set(figure(), 'Position', [500 300 800 500]);
plot(mag_abs,'k');
title('Absolute','Interpreter','latex','fontsize',14)
xlabel('Samples','Interpreter','latex','fontsize',14);
ylabel('Absolute Magnetic Field (Gauss)','Interpreter','latex','fontsize',14);
```

Listing E.4: scat.m Plots the magnetometer data

```
function scat(h)
    full=max(max(abs(h)))*1.05;
    full=round(full*10)/10;
    set(figure(), 'Position', [100 100 860 800]);

    subplot(2,2,1);
    scatter(h(1,:),h(3,:),'.k');
    title('XZ-plot','Interpreter','latex','fontsize',10)
    xlabel('X-sensor (Gauss)','Interpreter','latex','fontsize',10);
    ylabel('Z-sensor (Gauss)','Interpreter','latex','fontsize',10);
    grid on;
    axis([-full full -full full]);
    set(gca,'XTickMode','manual');
    set(gca,'XTick',(-full:.1:full),'YTick',(-full:.1:full));

    subplot(2,2,2)
    scatter3(h(1,:),h(2,:),h(3,:),'.k');
    title('3D-plot','Interpreter','latex','fontsize',10)
    xlabel('X-sensor (Gauss)','Interpreter','latex','fontsize',10);
    ylabel('Y-sensor (Gauss)','Interpreter','latex','fontsize',10);
    zlabel('Z-sensor (Gauss)','Interpreter','latex','fontsize',10);
    axis([-1 1 -1 1 -1 1]*full);

    subplot(2,2,3);
    scatter(h(1,:),h(2,:),'.k');
    title('XY-plot','Interpreter','latex','fontsize',10)
    xlabel('X-sensor (Gauss)','Interpreter','latex','fontsize',10);
    ylabel('Y-sensor (Gauss)','Interpreter','latex','fontsize',10);
    grid on;
    set(gca,'XTickMode','manual');
    set(gca,'XTick',(-full:.1:full),'YTick',(-full:.1:full));
    axis([-full full -full full]);

    subplot(2,2,4);
    scatter(h(3,:),h(2,:),'.k');
    title('ZY-plot','Interpreter','latex','fontsize',10)
    xlabel('Z-sensor (Gauss)','Interpreter','latex','fontsize',10);
    ylabel('Y-sensor (Gauss)','Interpreter','latex','fontsize',10);
    grid on;
```

```
    axis([-full full -full full]);
    set(gca,'XTickMode','manual');
    set(gca,'XTick',(-full:.1:full),'YTick',(-full:.1:full));
```

Appendix F

CD

The attached CD contains all software, firmware, figures, measurement data and the thesis in PDF format.