UNIVERSITY OF OSLO
Department of Physics

Development of a full-size low price Automatic Survey Vessel (ASV) for hydroacoustic work

Bendik S. Søvegjarto

May 15, 2015

Abstract

This thesis describes a pilot project for the development of an Automatic Survey Vessel (ASV) platform useful for hydroacoustic work. An inexpensive control unit, suitable for installation in an arbitrarily selected full-sized boat, has been developed. The control unit includes a motor control system, communication equipment, and a simple autopilot. This enables an operator to either remotely control the vessel during a survey or pre-program a route for the vessel to follow. Pre-programmed routes can be utilized to increase the efficiency of hydroacoustic surveys or to create complex driving patterns. The autopilot incorporates a GNSS-receiver and a self-developed tilt-compensated compass. The compass calculates the heading using sensor fusion from a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer. The motor control system, the compass and the autopilot have all been tested in either controlled- or field environments.

Acknowledgements

I would primarily like to thank my supervisor Dr. Helge Balk, who have been highly helpful and given great guidance while sharing his expertise. Furthermore, I want to express gratitude towards the following persons.

To my fellow students, Joakim Myrland, Johan Kleiberg Jensen, and Asbjørn Vinje, PhD candidates Atle Rustadbakken and Johan L. Tresvig, along with Associate Professor Torfinn Lindem for discussions, valuable suggestions and help with testing.

To Jan Kubečka and his fellow researchers affiliated with FishEcU, for welcoming me and helping me with fieldwork in the Czech Republic.

To Stein Lyng Nielsen and his colleagues at the electronics laboratory at UiO, for advise regarding electronic design.

To my father, Bernhard Søvegjarto, for financial support and advice regarding mechanical constructions.

Finally, to my partner, Magdalena Johansen, for helping me with the assembly of multiple printed circuit boards (PCBs), assisting me in numerous experiments, as well as supporting and encouraging me through the course of my master thesis.

Preface

My master thesis is naturally divided into separate parts, and is therefore written as multiple papers. The introduction covers the overview of the whole project.

I assume that the reader is familiar with basic concepts regarding electrical engineering, computer science and hydroacoustics.

Contents

List of Figures

List of Tables

List of Acronyms

AC	alternating current	4
AHRS	Attitude and Heading Reference System	
AP	Access Point	73
ASCII	American Standard Code for Information Interchange	
ASV	Automatic Survey Vessel	87
AWeX	Advanced Waveform eXtension	19
BJT	bipolar junction transistor	8
BLDC	brushless DC electric	4
BOOTP	Bootstrap Protocol	
CEP	Circular Error Probable	69
CGI	computer-generated imagery	
CNC	computer numerical control	
COTS	commercial off-the-shelf	3
CPU	central processing unit	
CRC	Cyclic Redundancy Check	40
CSS	Cascading Style Sheets	
DAC	digital-to-analog converter	12
DC	direct current	4
DFU	Device Firmware Upgrade	46
DGPS	Differential GPS	66
DHCP	Dynamic Host Configuration Protocol	73
DMP	Digital Motion Processor	43
EEPROM	Electrically Erasable Programmable Read-Only Memory	
EGNOS	European Geostationary Navigation Overlay Service	66
EMF	electromotive force	5
EMI	electromagnetic interference	5
ESD	electrostatic discharge	49

FPU	floating-point unit	84
FTDI	Future Technology Devices International	49
GCC	GNU Compiler Collection	44
GDB	GNU Debugger	69
GNSS	Global Navigation Satellite System	66
GNU	GNU's Not Unix!	
GPL	General Public License	103
GPS	Global Positioning System	69
GPT	General Purpose Transceiver	33
HDOP	horizontal dilution of precision	79
HP	Hewlett-Packard	33
HTML	HyperText Markup Language	84
HTTP	Hypertext Transfer Protocol	84
I^{2}C	Inter-Integrated Circuit	72
IC	integrated circuit	43
IETF	Internet Engineering Task Force	66
IGBT	insulated-gate bipolar transistor	15
IMU	Inertial Measurement Unit	
IO	input/output	
IP	Internet Protocol	72
IR	infrared	59
ISR	interrupt service routine	
JTAG	Joint Test Action Group	
LAN	local area network	
LED	light-emitting diode	
LFPAK	Loss-Free Package	23
MAC	Media Access Control	73
MCU	microcontroller unit	69
MEMS	microelectromechanical systems	42
MOSFET	metal–oxide–semiconductor field-effect transistor	8
MPL	Motion Processing Library	46
MSB	most significant bit	20
NLLS	Non-linear least squares	56
NMEA	National Marine Electronics Association	
NTP	Network Time Protocol	56

OSI	Open Systems Interconnection	
OSMC	Open Source Motor Control	15
OS	operating system	43
OpenOCD	Open On-Chip Debugger	
PCB	printed circuit board	89
PCM	pulse-code modulation	7
PHY	physical	
POE	Power-over-Ethernet	52
PPM	pulse-position modulation	7
PPP	Precise Point Positioning	67
PWM	pulse-width modulation	
QFN	Quad Flat No-leads	47
RADAR	RAdio Detection And Ranging	
RC	radio control	72
RF	radio frequency	
RMS	root mean square	
ROV	remote operated vehicle	
RPM	revolutions per minute	73
RTC	real-time clock	44
RTOS	real-time operating system	72
SBAS	satellite-based augmentation system	66
SBC	single board computer	44
SD	Secure Digital	75
TCP	Transmission Control Protocol	
TFTP	Trivial File Transfer Protocol	
TI	Texas Instruments	69
TVS	transient-voltage-suppression	26
TWI	Two Wire Interface	132
UART	universal asynchronous receiver/transmitter	
UDP	User Datagram Protocol	73
USB	Universal Serial Bus	75
US	United States of America	93
UUT	unit under test	
WAAS	Wide Area Augmentation System	66
WLAN	wireless LAN	81

Introduction

There is a growing need to study fish and water quality in lakes. This is reflected in political programs such as the EU Water Framework Directive [1]. Norway is through this directive obliged to examine the condition of all lakes over a certain size. This will not be economically feasible with the vast number of lakes matching this criteria and the available survey methods. These are largely based on the use of boats equipped with echosounders and other measuring equipment.

Hydroacoustic surveying on lakes is different from oceanic surveying because of the size of the lake, accessibility, ecology and economy. One must primarily use small boats and do the recordings at night. A small boat requires the driver to sit quietly to avoid reducing the quality of the recordings. Night work is also impractical, demanding, expensive and complicated to conduct in the dark because one must follow certain driving patterns for proper collection statistics. The echosounder will only see a limited portion of the water, since the sound beam is formed like the cone from a flashlight. At shallow lakes one can increase the coverage by doing longer surveys or use more boats. Another option is to direct the sound beam horizontally. Horizontal driving is however subjected by a number of problems related to reflection, refraction, determination of the orientation of the fish in the water and converting echo strength to fish size.

Many of these issues could be solved with a larger number of affordable automatic and self-steering survey vessels. Several boats can be launched at night and retrieved in the morning. We get greater coverage per task force and researchers will be able to analyze data by day instead of driving around the lakes at night. Much research remains on horizontal use of echosounders before this method can provide usable results. An automatic boat will make a significant contribution to this research in terms of providing a stable quiet platform with possible pre-programmed or dynamic driving patterns that is difficult to emulate with manned boats.

The main task in this project is the development of such a vessel. There already exists a few such boats, but they are either developed for oceanic use, with a size and price [2] which makes them totally unsuitable for lake research, or they are small model boats [3] unable to carry the necessary research equipment and energy to conduct a longer survey.

We therefore want to develop an Automatic Survey Vessel (ASV) platform. We will in practice accomplish this by developing a control unit. It should be possible to install this control unit in an arbitrarily selected boat, powered by two arbitrarily selected electric outboard motors. This will enable research institutions to choose boats and motors they already have available, and which are adapted to the actual lake.

The control unit in this project will include a motor control system, communication equipment and a simple autopilot. The autopilot uses a Global Navigation Satellite System (GNSS)-receiver and a self-developed compass for navigation.

Thesis outline

The development of the motor control system with simple communication is covered by the paper «Construction of a remote controlled work platform for hydroacoustic work» on page 3. The development of the compass is covered by the paper «Construction of an Attitude and Heading Reference System (AHRS)» on page 41. The development of the simple autopilot is covered by the paper «Construction of a primitive autopilot for hydroacoustic work» on page 65.

Paper 1

Construction of a remote controlled work platform for hydroacoustic work

Abstract

This paper presents the construction and implementation of the steering mechanism and motor control of a remote controlled work platform for hydroacoustic work. Multiple high current brushed DC-motor controllers, based on H-bridge circuits, have been developed. After a disappointing attempt with a commercially available H-bridge driver, a custom H-bridge driver has shown good results. The final system has, at multiple occasions, proved it can remote-control full-sized boats. The driver can also be extended for further automation.

Keywords: H-bridge, ROV, motor control

1.1 Introduction

Mobile hydroacoustic surveys in small boats requires the crew to sit quietly to avoid reducing the quality of the recordings. By using a remote controlled vessel, an operator can move freely on shore while conducting the survey at a distance.

Further development can extend the remote control functionality, and let the vessel follow pre-programmed patterns, making it even more useful for researchers. Finally a fully autonomous system can be built, based on the foundation developed in this project.

This paper will present the development of a motor control system with remote-control functionality. One of the cornerstones of the project was to build our system using only commercial off-the-shelf (COTS) parts. This later enables us to build a fleet of survey vessels, and makes it inexpensive for other institutions to adopt. The system will also build upon boats and motors the user already may have available.

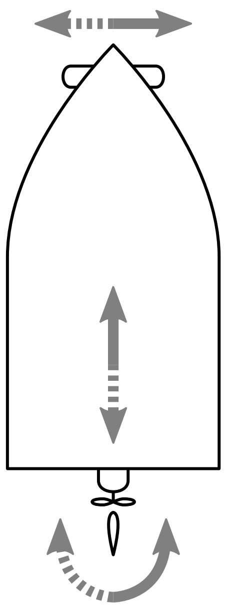
Similar remote-controlled systems have already been created, but they are too small [4] to carry the necessary research equipment or too expensive [2].

1.1.1 Motor placement

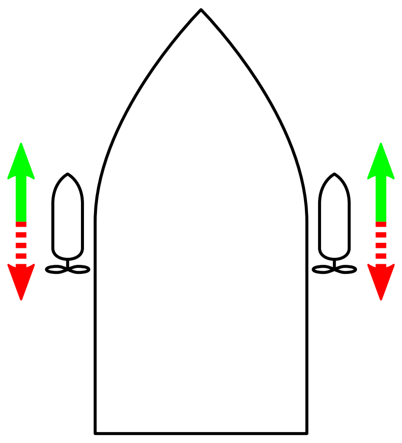
There are multiple possible setups that can be used to steer a boat, and it was especially two different setups we were considering.

The first alternative is to control the throttle of a rear mounted outboard motor, together with a steering mechanism. The steering mechanism could be the turning of a rudder, turning of the whole motor or the throttle of a thruster[1]

[1]A thruster is a propulsion device mounted across the bow or stern of a boat to increase its maneuverability



(a) Motor placement using a rear mounted engine, together with a thruster or a rudder.



(b) Motor placement using two engines, one mounted on each side.

Figure 1.1: Motor placement options.

mounted across the boat at the bow. This configuration is shown in figure 1.1a.

The second alternative is to control the throttle of two outboard motors. With one mounted at the port side and one mounted at the starboard side, as shown in figure 1.1b.

The first alternative would either have required the use of a servo motor to turn the rudder or the engine, or an expensive thruster. The second alternative, on the other hand, can be accomplished without any moving parts (besides the movement of the propeller). The second motor placement alternative will also make it possible to «turn 360 ° on-the-spot».

1.1.2 Motor type

The main motors needs to be powerful enough to drive the boat, while being easy to control. We were considering the use of either a fuel/petrol engine or an electric engine.

Electric engines are often less complicated to control compared to a fuel/petrol engine. This is because it is possible to control the thrust by simply controlling the electric current running through them. A brushed direct current (DC)-motor can also with ease be used to drive a boat in reverse. Switching the polarity of the current will result in the movement direction of the propeller to be switched as well. A fuel/petrol engine requires mechanics to achieve this ability. On the other hand, an electric engine usually provides less thrust than a fuel/petrol engine. Since the vessel will only move at slow speed, and in low (water-)current environments, we still believe an electric motor is sufficient for our requirements.

Other possible electric motor types besides brushed DC-motors, are alternating current (AC) and brushless DC electric (BLDC) motors. AC motors were not considered because of the lack of an AC source in the vessel. BLDC motors were considered as they have multiple good features.

BLDC motors do not have any mechanical connection between the stator and the rotor. They rely on multiple stator coils being alternatively magnetized at the correct rotor angle to drive the shaft. These motors require less maintenance because of the lack of brushes, gives high torque regardless of speed, and overall higher efficiency. Brushed DC motors have arcing by their brushes which generates electromagnetic interference (EMI), while BLDC do not. Unfortunately they require an advanced motor controller, as the timing of when the coils should be driven must be exact and depends on the rotor angle. The angle of the rotor can be found using hall effect sensors, or sensing the back-electromotive force (EMF) from the undriven coils.

Despite some of the good characteristics with a BLDC motor, we chose to use brushed DC motors. The reason being high availability of inexpensive brushed DC outboard engines and the requirement of a much simpler motor controller. Fuel/petrol engines were taken out of consideration, because they are more complex to control, and we did not need the high thrust they can deliver.

We also selected the second motor placement alternative, with one engine mounted on each side of the boat. Together with electric engines, this combination requires no movable parts (except for the propeller).

We found two inexpensive 12 V electric outboard engines with a pulling force of 15.4 kg, shown in figure 1.2a. The engines had a purchase price of about 1500 NOK each, and are available from Biltema (product nr. 25235). Any 12 V brushed DC engine the user already may have available should be suitable (as long as the current draw is limited). We were also considering Endura C2 from Minnkota, but chose the one from Biltema because it was easier for us to source. Our engines are rated to draw up to 32 A of electric current, but the motor control system is designed to handle 40 A.

If two different engines are used, their difference in thrust can be compensated in software. This is, however, currently not implemented.

The inside of the Biltema outboard engine is shown in figure 1.2b. The engine has a control mechanism in the handle where multiple speeds can be selected. The handle can also set the engine in reverse. We considered bypassing this control mechanism, connecting straight to the motor. After reverse-engineering the connections inside the control system, however, we found that we have a direct connection to the motor when the maximum speed is selected. Our motor control system can therefore connect to the battery terminals of the engine, without modifying or breaking the warranty of the device.

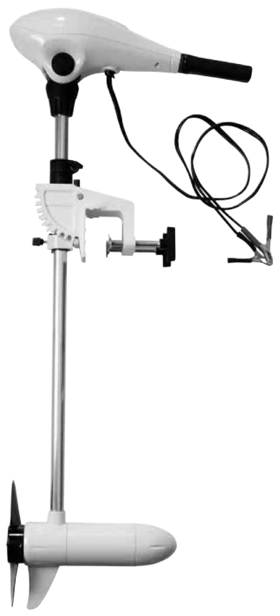
To control the direction and thrust of the engines, we were in need of a high current brushed DC-motor controller.

1.1.3 H-bridge

With the requirement of running the electric engine both forward and backward, we chose to use an H-bridge. This will also make it possible to adjust the thrust of the engine. An H-bridge solves these challenges by directing the current to run through the motor in the wanted direction. We can control (pulse-width modulate) the amount of power the motor receives by turning this current rapidly «on» and «off».

The principal operation of an H-bridge is covered by section «H-bridge operation» on page 8. The implemented operation of our H-bridge is covered by section «Second H-bridge drive modes» on page 20.

We considered buying a commercial H-bridge motor control system, but chose to develop our own. The reason being lower cost and full control of the whole system. Our motor control system consists of an H-bridge controller, an H-bridge drive-stage and an H-bridge. In order to form a complete remote controlled motor control system, a radio control (RC)-receiver and a steering unit is included as well. Two attempts have been conducted; the first described in section «First attempt on H-bridge construction» on page 12 and the second in section «Second attempt on H-bridge construction» on page 19.


(a) Complete view of the electric outboard engine used.

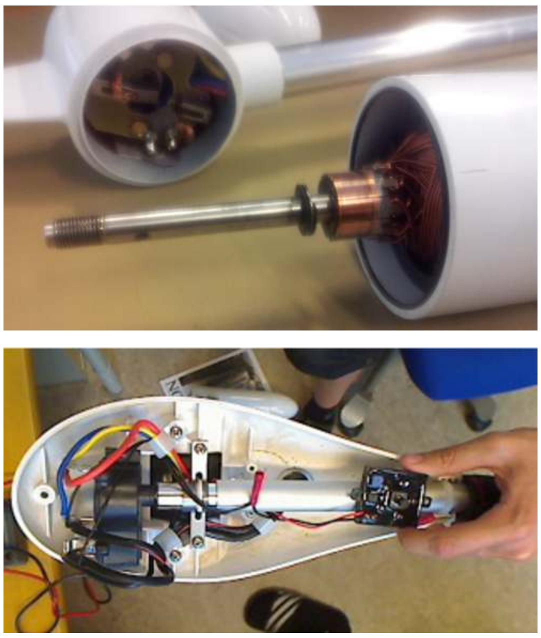

(b) Inside view of the handle and motor.

Figure 1.2: Electric outboard engine from Biltema.

(a) Turnigy 9X RC-transmitter.

(b) RC-receiver.

Figure 1.3: Photographs of RC-communication equipment.

1.1.4 Communication

We need a way for the operator to send wireless commands to the boat. For the remote control operation described in this paper, the commands will be simple control signals to propel and steer the vessel. We considered different radio communication systems, such as IEEE 802.11 wireless LAN (WLAN) or hobby-radio control (RC) with pulse-position modulation (PPM) or pulse-code modulation (PCM) encoding.

It became apparent that using a standard RC-servo control signal, commonly used by hobby enthusiasts, would be a good choice. This system is widely used, simple to implement and would fulfill our requirements of long range, low price and sufficient bandwidth for sending simple control signals. It does not include any form for encryption or authentication, but security regarding control over the vessel is currently not vital as the vessel will be strictly operated under human supervision.

Further development, such as the implementation of an autopilot (paper 3), may use another radio communication system with encryption and higher bandwidth. The RC-control signal could still be used as a redundant communication channel, to override any automatic steering system and regain manual control over the vessel.

We chose a 9-channel 2.4 Ghz RC-transmitter named Turnigy 9X, with a matching 8-channel receiver[2] .

This RC-transmitter was chosen because it *a*) is inexpensive; *b*) is reliable (less interference, compared to a 27 MHz system); *c*) employs frequency hopping; *d*) has good range; *e*) is easily available (under multiple brand names); and *f*) has alternative firmware. The transmitter contains an Atmel ATmega64 8-bit microcontroller unit (MCU), two sticks, three potentiometers, seven switches and a 128x64 dot display. Several developers have created custom and free[3] firmwares for this unit, e.g. er9x [5] and th9x [6]. Since the stock firmware already loaded into the transmitter fulfilled our needs, we did not use any of these custom firmwares. One of the alternative firmwares may, however, be customized for the project in the future. For instance to show information about how to use and/or setup the vessel on the display, or manage the switches and potentiometers to give additional control possibilities. The RC-transmitter is shown in figure 1.3a.

Each of the 9-channels is connected to an output on the receiver, and may have different purposes depending on the setup of the transmitter. The setup of our transmitter sends the position of the left stick to the first channel and the position of the right stick to the second. None of the other channels were used. The RC-receiver is shown in figure 1.3b.

The output of the RC-receiver is a standard servo control signal with a constant frequency of 50 Hz, but varying pulse-width depending on the position of the stick on the transmitter. A pulse-width of 1.5 ms corresponds to a stick in its center position, while 1 ms and 2 ms corresponds to minimum and maximum stick position respectively. These pulse-widths are shown in figure 1.4.

The user may use any RC-system he or she has available, as long as it is outputting these de facto standard servo signals.

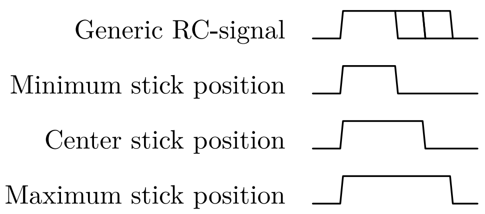
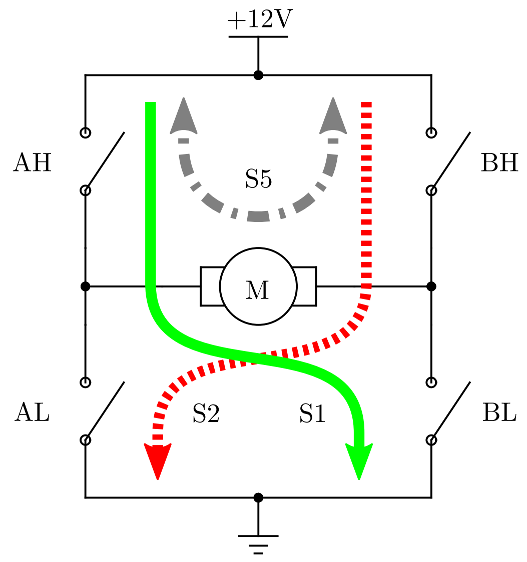


Figure 1.4: Illustration of RC-servo signals. Different pulse-widths corresponds to different stick positions.

[2]Also sold under other brand names, such as Eurgle, FlySky and Imax. Available from HobbyKing at about 50 EUR, `http://www.hobbyking.com/hobbyking/store/__8991__Turnigy_9X_9Ch_Transmitter_w_Module_8ch_Receiver_Mode_1_v2_Firmware_.html`

[3]as in free speech



(a) Drawing of H-bridge with currents.

(b) H-bridge operating states.

#	H-bridge state	AH	AL	BH	BL
S1	Motor runs cw	on	off	off	on
S2	Motor runs ccw	off	on	on	off
S3	Motor runs free	off	off	off	off
S4	Motor brakes	off	on	off	on
S5	Motor brakes	on	off	on	off
S6	Shoot-through	on	on	off	off
S7	Shoot-through	off	off	on	on
S8	Shoot-through	on	on	on	on

Figure 1.5: Illustration and table for H-bridge principal operation.

1.2 Theory

1.2.1 H-bridge operation

An H-bridge is a circuit topology where four switches and a load are connected in the shape of an «H», as shown in figure 1.5a. These switches can either be mechanical or electronic, and can turn each leg of the H-bridge «on» or «off».

An H-bridge is operated in different states, by setting its switches in different configurations (table in figure 1.5b). By turning the switches AH and BL «on», while AL and BH are «off», a current is flowing through the motor as indicated by the green arrow in figure 1.5a. This makes the motor shaft turn clockwise, as shown in table 1.5b. Subsequently AL/BH may be «on», while AH/BL are «off», to make the motor shaft turn counter-clockwise[4] (red arrow). If all the switches are «off», the motor will run free. If the top (or bottom) switches are «on», the motor will brake (gray arrow).

Some configurations will result in a short circuit between the power rail and ground, and must be avoided by all means. This is called a shoot-through, and happens if two switches on the same side are «on» simultaneously. To prevent this, the H-bridge controller must make sure there is a time delay between turning two transistor on the same side «on».

Transistors are often used as the electronic switches in an H-bridge, such as two pairs of NPN and PNP bipolar junction transistors (BJTs) or two pairs of P- and N-channel metal–oxide–semiconductor field-effect transistors (MOSFETs). Since these are complementary, they will turn «on» and «off» inversely of each other. Alternately, just N-channel MOSFETs may be used, as P-channel MOSFETs have much higher $R_{ds_{\mathrm{on}}}$, lower current rating and higher price. This requires, however, a drive-stage which can supply a voltage higher than the motor's power supply to turn the N-channel MOSFETs «on». For a N-channel MOSFET to turn «on», it must have a high potential on its gate with respect to its source.

The states listed in table 1.5b shows how the H-bridge is controlled during static operation. E.g. when all the current is flowing continuously in the wanted direction and the motor is running

[4]What direction the motor is turning (cw/ccw) is of course depending on which way its wires are connected.

at full speed. To change the speed of the motor, the drive voltage can be altered, as the speed of the motor is proportional to the drive voltage. A higher voltage will make the motor draw a larger current and consume more power, and vice versa.

Another solution is to turn the motor rapidly «on» and «off». It is easier to turn the drive current «on» and «off», than to change the drive voltage, since we already have the necessary switches in place. The ratio between the «on» and «off» time of the drive current will set the speed, as the motor receives different amounts of energy. To control the ratio, the switches are driven with a pulse-width modulated PWM signal. A PWM signal has constant frequency, but varying pulse-width[5], effectively changing the ratio between «on» and «off», as shown in figure 1.6. The motor is then duty-cycle controlled.

Further discussion about how the H-bridge drive modes is actually implemented is covered by section «Second H-bridge drive modes» on page 20. A more detailed analysis about H-bridge operation is covered by a series of articles beginning with [7].

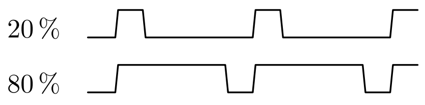


Figure 1.6: Illustration of (two) PWM signals. Top signal has a pulse-width (and thus ratio) of 20 %, while the bottom signal has a pulse-width of 80 %.

1.2.2 Calculated heat dissipation in MOSFETs

The current flowing through the MOSFETs in the H-bridge will dissipate heat in two ways.

When they are in their saturated region (completely turned on), their resistance between drain and source (R_{ds}) will lead to a constant power consumption depending on the current running through them. This power is given by Ohm's law (1.2) and the electric power equation (1.3), which yields equation (1.1).

$$P_{\text{on}} = R_{ds} \times I_{\text{max}}^2 \tag{1.1}$$

where P is electric power, R is resistance and I is current.

Ohm's law is given as

$$U = R \times I \tag{1.2}$$

where U is the voltage, R is the resistance and I is the current.

When the MOSFETs are changing between their «on»- and «off»-state on the other hand, they dissipate power during the switching-transition. We want to find an expression of this dissipated power. To disregard the power consumed while the MOSFETs are saturated, we look at a MOSFET with $R_{ds} = 0$. The current through the MOSFET while switching is shown in figure 1.7 and the voltage over it is shown in figure 1.8. By using the electric power equation (1.3) we get the power shown in figure 1.9.

Electric power is given as

$$P = U \times I \tag{1.3}$$

where P is electric power, U is voltage and I is current.

Figure 1.9 shows us the power dissipated as heat in one switching-transition. We can sum this power by solving the integral of the power over time, and get the energy lost as heat by each switching-transition. The length of each switching-transition is denoted by the time Δt.

Energy lost during a switching-transition is thus given as

$$E = \int_0^{\Delta t} P(t)\,\mathrm{d}t \tag{1.4}$$

where E is energy, P is electric power and t is time.

This energy is converted to heat twice in each period. First on the rising edge and then on the falling edge. We can now determine how many times a second this energy is converted to heat. This is where the switching frequency

[5]The RC-signal described in section 1.1.4 is thus also a PWM signal.

gets involved. The energy from two switching-transitions can be multiplied with the frequency, and will then give the time-averaged energy per second. This is the definition of power and we have thus found an expression for the dissipated heat caused by the switching.

Average power lost in heat due to all switching-transitions is therefore given as

$$P = f \times 2E \tag{1.5}$$

where P is electric power, f is the switching frequency and E is energy.

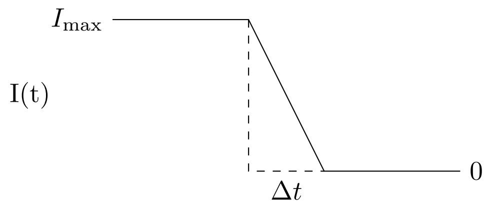


Figure 1.7: Current through a MOSFET during a switching-transition.

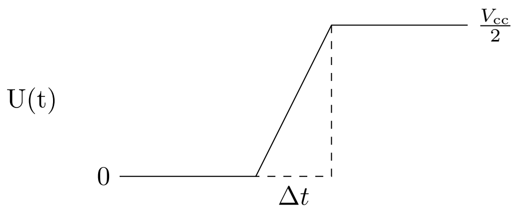


Figure 1.8: Voltage over a MOSFET during a switching-transition.

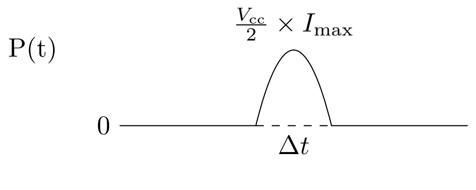


Figure 1.9: Power dissipation in a MOSFET during a switching-transition.

We can find the heat loss if we apply a linear approximation for the current and voltage during the switching-transition (as shown in figure 1.7 and 1.8).

The current during the switching-transition can be described as

$$I(t) = I_{\max} - \frac{I_{\max}}{\Delta t} t \tag{1.6}$$

which can be seen directly from figure 1.7.

Similarly, the voltage can be described as

$$U(t) = \frac{\Delta t}{\frac{V_{cc}}{2}} t = \frac{2\Delta t}{V_{cc}} t \tag{1.7}$$

$\frac{V_{cc}}{2}$ originates from the total voltage (V_{cc}) being divided amongst the two MOSFETs.

By combining equation (1.6) and (1.7) with equation (1.3) we get equation (1.8).

$$P(t) = \frac{2\Delta t}{V_{cc}} t \left(I_{\max} - \frac{I_{\max}}{\Delta t} t \right) \tag{1.8}$$

The energy lost during one switching-transition (E) is, as earlier discussed, the power (P) lost over the time Δt, expressed in equation (1.9).

$$E = \int_0^{\Delta t} \frac{2\Delta t}{V_{cc}} t \left(I_{\max} - \frac{I_{\max}}{\Delta t} t \right) \mathrm{d}t \tag{1.9}$$

After solving the integral in equation (1.9) we get equation (1.10).

$$E = \frac{I_{\max} \Delta t^3}{3V_{cc}} \tag{1.10}$$

Using equation (1.5), the average power lost in heat due to all switching-transitions is then given by equation (1.11).

$$P = \frac{2f \times I_{\max} \Delta t^3}{3V_{cc}} \tag{1.11}$$

These calculations show that the switching loss increases with the switching frequency (f), current ($I_{\max}$) and transition time (Δt). The current is governed by the load, and the transition time is determined by the choice of MOSFETs and driver. The switching frequency can be adjusted, and should be kept low to minimize switching losses.

There is also switching loss involved with the charge and discharge of parasitic capacitances within the MOSFET. Another new MOSFET model for estimating switching loss is presented in [8].

1.2.3 Calculated copper track width

When routing printed circuit boards (PCBs) for the H-bridges, a suitable width for the tracks conducting the drive current to the motors must be found. If the track are too narrow, the copper laminate will get too hot and burn. By setting a limit on dissipated power (e.g. 1.5 W) for a given length (50 mm), we can calculate the needed track width. The dissipated power will increase with the track length, so the width needed to keep a constant temperature is usually independent of track length. The allowed dissipated power is dependent on the ambient temperature, allowed temperature rise, and thermal resistance between the track and the environment. IPC-2221 [9] can be used to find these factors.

The resistance R of a wire with cross sectional area A, length l and resistivity ρ is given by Pouillet's law

$$R = \rho \frac{l}{A} \tag{1.12}$$

Cross sectional area A is given as the width w of the track multiplied with its height h (thickness). We can then solve Pouillet's law (1.12) for w and get an expression for the width w needed to have a resistance R. We can use equation (1.1) to get an expression for the allowed resistance R to not exceed the power P. By combining Pouillet's law (1.12) with equation (1.1) we get equation (1.13).

$$w = \frac{\rho I^2 l}{hP} \tag{1.13}$$

where w is track width, h is track thickness, l is track length, P is allowed dissipated power, ρ is resistivity and I is current.

The resistivity ρ of copper is 17 nΩm [10, p. 658]. The thickness h of the outer copper sheets of a standard PCB is 35 μm. With a current of 40 A, and a track length of 50 mm, we need a minimum width of 25.9 mm to have a power dissipation below 1.5 W. A power dissipation of 1.5 W over 50 mm with this width corresponds to a temperature rise of about 25 ° (very high) for external layers in air and an ambient temperature of 25 ° [9].

An interesting result of Pouillet's law is that (when the thickness is fixed) the resistance of any square (width = length) will be equal, regardless of its size. A 1 mm × 1 mm square has the same resistance as a 1 m × 1 m square. The resistance will only change when the length-to-width ratio is altered.

1.3 Material and methods

Two attempts have been made to create a full motor control system. Both motor control systems would have incorporated *a*) one RC-receiver for communication; *b*) one or two steering controller(s); *c*) one or two H-bridge controller(s); *d*) two H-bridge drive-stages; and *e*) two H-bridges.

The RC-receiver is responsible for receiving the stick position of the remote control unit used by the operator. The steering controller is responsible for translating the stick position to driving commands suitable for the H-bridge controller. The H-bridge controller is responsible for controlling the H-bridge drive-stage, so the correct transistors are turned «on» and «off», and without creating a shoot-through condition. The H-bridge drive-stage is responsible for driving the gate of every transistor in each leg of the H-bridge, with the correct voltage and with enough current to minimize the transition time (Δt). The H-bridge is responsible for conducting the drive current to one of the motors in the wanted direction. H-bridge driver is the name for the H-bridge controller and H-bridge drive-stage combined.

The first attempt was designed around the HIP4081 integrated circuit (IC) which is both an H-bridge controller and a drive-stage. This IC was placed on a PCB together with an H-bridge. The steering controller was first made using only analog components. The full motor control system, using these components is shown in figure 1.10a. Two analog control signal decoders are used as steering controllers. Both are needed to control the boat, with each connected to an H-bridge (with controller and drive-stage). Each H-bridge will then drive one of the electric engines. The left stick on the RC-transmitter controls the left engine, while the right stick controls the right engine.

Since the analog steering controller was troublesome (described later), a new digital control signal decoder was developed. By using the new digital control signal decoder as the steering controller, both sticks on the RC-transmitter can be combined to create a more sophisticated steering mechanism (described in section 1.3.1). A block schematic of the new motor control system (still using the first H-bridge) is shown in figure 1.10b.

After problems with the first H-bridge solution were encountered (described in section 1.3.1), a new H-bridge drive-stage was constructed together with a new H-bridge. This new systems is based on the previous digital steering controller, but incorporates the H-bridge controller as well. The new motor control system is shown in figure 1.11.

1.3.1 First attempt on H-bridge construction

The first attempt on an H-bridge construction started by creating an analog control signal decoder, which constitutes the steering controller (link between the RC-receiver and the H-bridge).

Analog steering controller

To get the stick position of the RC-transmitter, the pulse-width from the RC-receiver needs to be measured. This should in turn drive the H-bridge accordingly.

My first attempt was to accomplish this using only analog components. I used the circuit listed in appendix C based on a circuit from [11], and created the PCB shown in figure 1.12. This board would connect the RC-receiver to an H-bridge, by converting the servo-signal to two PWM pulses needed to drive the H-bridge. One to drive the motor forward, and one to drive it backward.

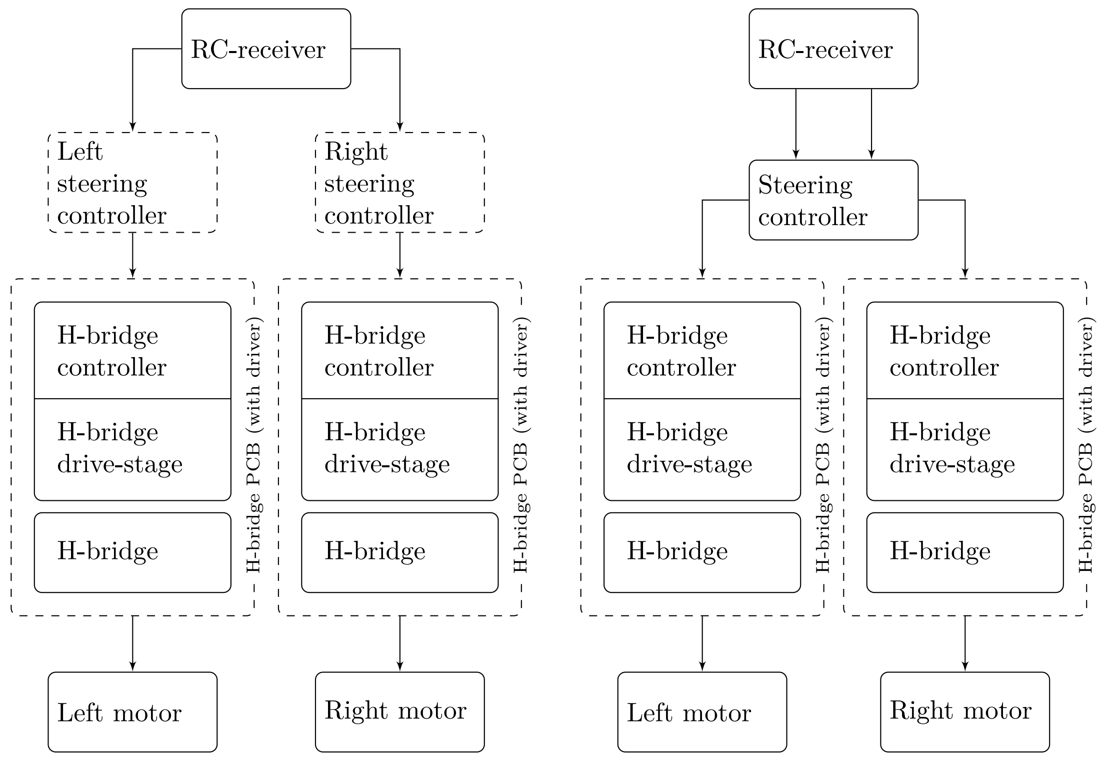
The circuit works by charging capacitor C2 and C3 with a buffered servo-signal. The charged voltage is sampled when the pulse ends. This sampled voltage is compared with U2A and U2C, against a triangle wave generated by U2D. This results in two PWM pulses which can drive the H-bridge. One is output when the stick is positioned forward and the other when the stick is positioned backward.

Digital steering controller – proof of concept

The analog steering controller has a non-linear relation between the pulse-width and the voltage to the comparator. It was also difficult to adjust the timing so the analog switches opened and closed when they should. For these reasons, I began looking for another way to measure the pulse-width.

Multiple options were considered. One solution was to integrate the pulse with an operational amplifier and a capacitor to create a voltage proportional to the pulse-width. Another solution was to let a counter run for as long as the pulse is high and send this digital output to a digital-to-analog converter (DAC). A circuit with many analog components often requires a lot ICs, which takes a lot of time to build and give many sources of error. I realized that the problem of reading the servo signal and output it to the H-bridge is straightforward to accomplish using a MCU. Everything can be done in the same IC – measuring the pulse-width with an internal counter, possibly communicate with another MCU, and generating a PWM drive signal along with a direction bit.

I used an MCU named ATtiny85 by Atmel which I had by hand, to create a proof of concept for this solution. This MCU has a counter and PWM output capabilities. I wrote the program listed in appendix D. The program receives an interrupt on the edges of the servo signal, and resets the counter if it is rising or reads the counter if it is falling. Depending on the value of the counter, a PWM signal will be output along with a direction indicator. The system was tested as shown in figure 1.13, and gave promising results, as it could successfully dim a LED depending on the stick position of the RC-transmitter.



(a) Using the analog steering controller. (b) Using the digital steering controller.

Figure 1.10: Block schematic of a full motor control system using a pair of the first H-bridges, with either analog or digital steering controller. Dashed boxes indicates self-made PCBs.

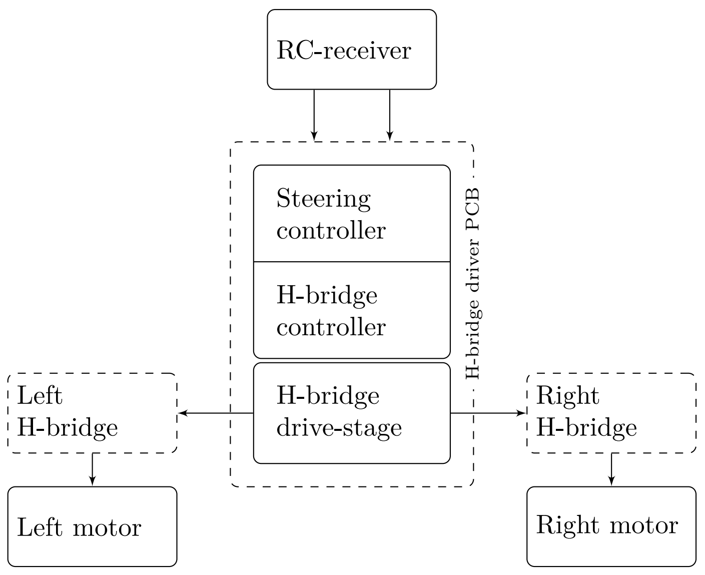
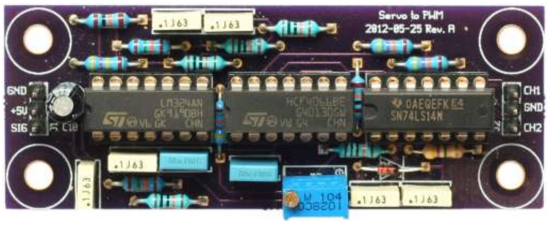
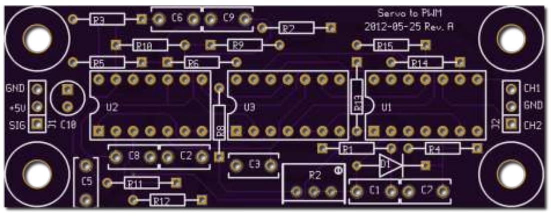


Figure 1.11: Block schematic of a full motor control system using the second H-bridge. Dashed boxes indicates self-made PCBs.



(a) Photograph of an assembled PCB of the analog steering controller.



(b) CGI of the analog steering controller PCB.

Figure 1.12: Images of the PCB for the analog steering controller.

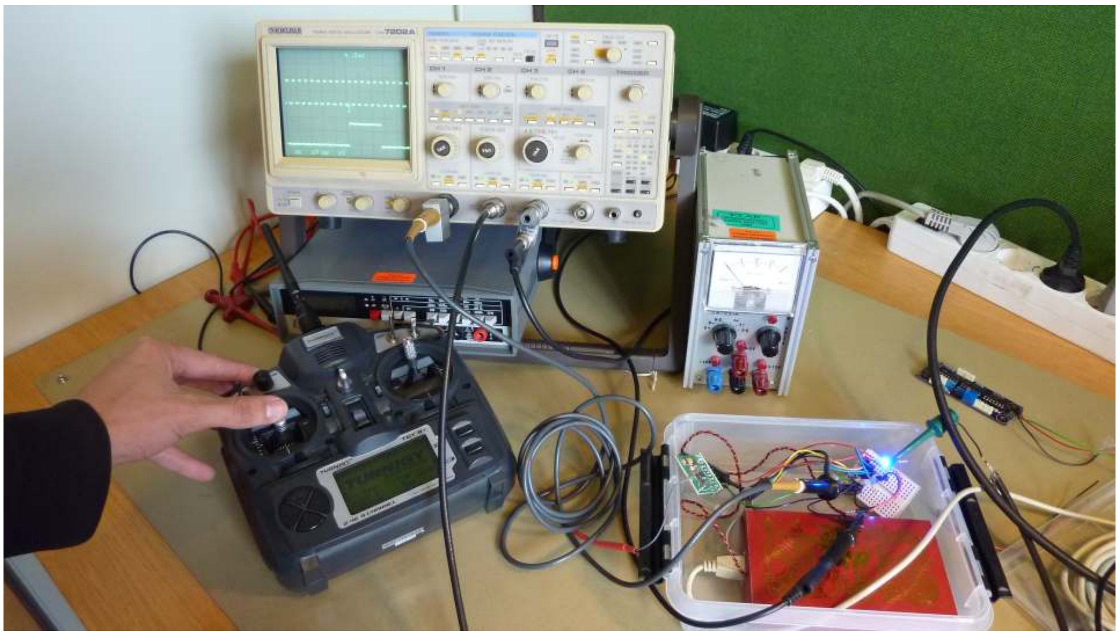


Figure 1.13: Test of the prototype digital steering controller running on ATtiny85.

Digital steering controller – dual channel

Some of Atmel's other MCUs supports a special interrupt called Input Capture. This interrupt have the ability to timestamp events, and can give out information such as frequency or pulse-width directly. ATtiny10 has Input Capture and was considered for the job. With its one Input Capture channel it could have done the same job as the analog control signal detector. In the end an ATxmega128A1 was chosen because it has multiple Input Capture channels, making it easier to create a system which incorporates two control sticks.

Development of the ATxmega128A1 code was originally done on a breakout board containing an ATxmega128A1 MCU, a crystal, and other necessary components. The breakout board can be seen in the middle of figure 1.15. The code was initially developed in Atmel Studio, but Atmel Studio was eventually replaced by a stand alone compiler (GNU Compiler Collection (GCC)), build utility (make), flasher/programmer (avrdude) and a text editor (vim), for a more transparent workflow. I wrote the program shown in appendix E. This program uses Input Capture interrupts to find the pulse-width of each channel. This pulse-width is scaled and modified with a dead band (around center) to find the stick position.

The program then calculates the thrust to each engine by combining the stick position of both sticks. The total thrust of both engines is controlled by the left stick. The difference between the thrust to each engine is controlled with the right stick. In other words, the left stick sets the speed of the boat, while the right stick sets the direction. The steering mechanism is presented in table 1.1.

It is also possible to choose a more primitive steering mechanism, by the flip of a switch. Each stick would then control the thrust of one motor, and give the same behavior as using two single steering controllers.

Finally it creates four PWM signals. Two for the left motor (forward and backward), and two for the right motor.

The switching (PWM) frequency was set to 1 kHz as this is high enough for the motor to run smoothly, while low enough to avoid excessive switching losses (as described in section 1.2.2). The chosen switching frequency lies in the audible spectrum for humans, but we do not consider this a problem since the vessel will usually be located away from people. It may be necessary to change the switching frequency, to stop its harmonics from interfering with (the frequency of) the echosounder used during a survey. The switching frequency can easily be altered in software.

First H-bridge controller and drive-stage

After researching different H-bridge constructions, multiple options were found. It is possible to buy all assembled H-bridge-motor drivers as modules or as monolithic ICs with H-bridge controller, drive-stage and power transistors all embedded. ICs with just the controller and drive-stage are also available.

As described in 1.1.3, we chose not to buy an all assembled H-bridge-motor driver module (assembled PCB with enclosure). ICs with everything included were then considered. The SN754410 is an all-in-one H-bridge and can drive up to 36V at 1A continuous output current, but this is far from enough for our application. NCV7729 is a similar IC from ON semiconductor that can withstand up to 8 A, but this is still is not enough.

Due to the high current required, it was decided that discrete power transistors were necessary. A drive-stage for these transistors, as well as an H-bridge controller were still needed.

Multiple pre-made H-bridge driver ICs were found, e.g. LT1162 and HIP4081. They have embedded logic for driving top and bottom transistors, protection against shoot-through conditions and generation of the high voltage needed to turn «on» the top N-channel MOSFETs. The HIP4081 from Intersil was finally chosen because it is widely used and have showed good results in the Open Source Motor Control (OSMC)-project [12].

First selection of power transistors

We needed a minimum of four power transistors which each should handle the high currents needed to drive the electric engines. The possible transistor types were BJT, MOSFET or insulated-gate bipolar transistor (IGBT).

An IGBT is a kind of cross between a MOSFET and a BJT, as it is voltage controlled like a MOSFET, but has the power driving capabilities of a BJT. It can typically withstand much higher voltages (> 1000 V), has lower R_{on} and less stray capacitances than a MOSFET.

BJTs were not considered because of their poor switching performance, such as longer turn-on and turn-off times. We chose MOSFETs over IGBTs, because we do not need the (very) high voltage and (very) high power capabilities of the IGBTs. MOSFETs will then have a larger selection at our specifications of 12 V and about 40 A.

The FDP3651U, a 100 V 80 A N-channel MOSFET, was considered among others, but was finally rejected in favor of IRF1404z. IRF1404z has a lower $R_{ds_{on}}$ of only 3.7 mΩ, compared to 18 mΩ for FDP3651U. IRF1404z can withstand a continues current of 75 A at 10 V and pulsed currents up to 750 A. Maximum

Table 1.1: Steering mechanism. Vessel behavior at various stick combinations.

Left stick ↕	Right stick ↔	Vessel behavior
Forward	Center	Drive straight forward
Forward	Left	Drive forward, while turning left
Forward	Right	Drive forward, while turning right
Backward	Center	Drive straight backward
Backward	Left	Drive backward, while turning left
Backward	Right	Drive backward, while turning right
Center	Center	Stand still
Center	Left	Turn counter-clockwise on the spot
Center	Right	Turn clockwise on the spot

V_{ds} is 40 V. We chose to use a TO-220 through-hole package for easy attachment to a heatsink.

Heatsink

To cool the discrete transistors, we initially thought that a heatsink was required. I chose SK 61/100 SA from Fischer Elektronik, because it has mounting brackets and room for the PCB in the middle, as shown in figure 1.17a.

Protection circuitry

Extra flyback diodes are connected in addition to the internal suppression diodes. These diodes protects the transistor from the sudden voltage spike the inductive load (motor) induces when its current is suddenly removed. They provide a path for the energy stored in the motor to escape back to the power supply, if no other path is available.

An RC-snubber circuit (consisting of a resistor and capacitor in series) connected in parallel with the load, was considered to be incorporated as well to help suppress voltage transients. In the end, this was not found to be required.

Design and production of first H-bridge

A PCB based on the HIP4081 was designed and manufactured. The schematic is shown in appendix F, the PCB in figure 1.14 and the assembled PCB in figure 1.16 and figure 1.17b (with heatsink).

The PCB was designed using Zuken Cadstar, and etched in-house at the electronics laboratory. After etching, a thin coat of SUR-TIN[6] was applied to help soldering, as visible in figure 1.16. A lot of solder paste was applied to make the tracks thicker, before it was soldered in a vapor oven.

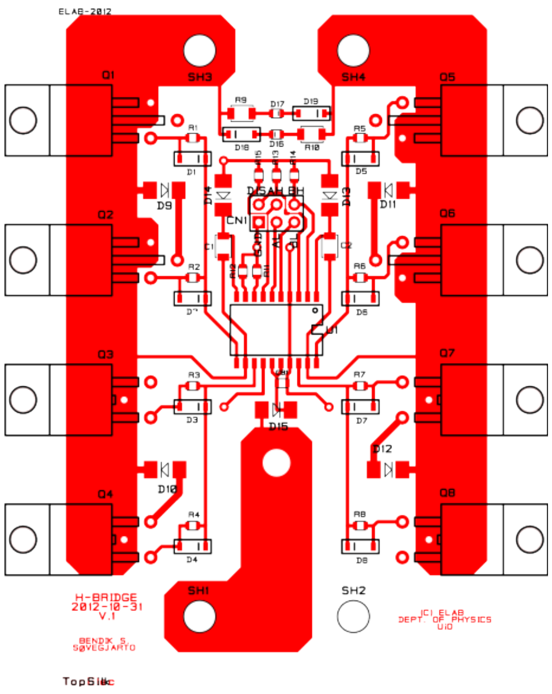


Figure 1.14: PCB layout for the first H-bridge.

[6]Electroless immersion tinning system

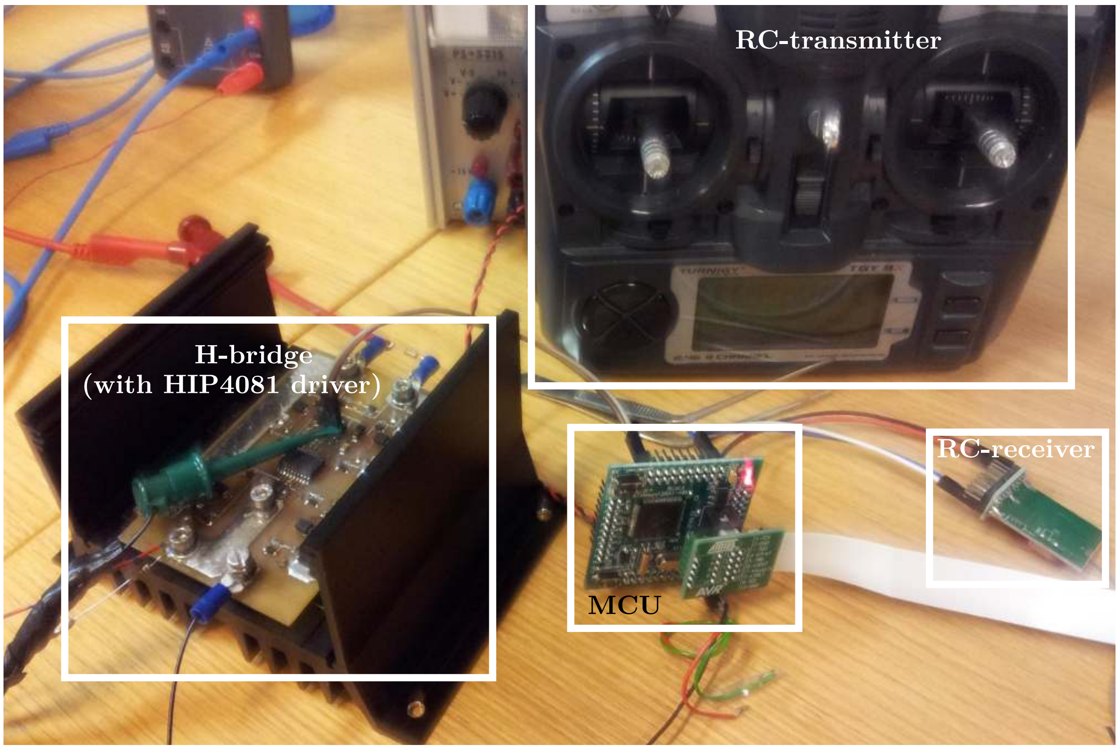


Figure 1.15: Test setup for the first H-bridge.

The PCB incorporates eight MOSFETs, with two connected in parallel for each leg in the H-bridge. This was done to lower the effective $R_{ds_{on}}$, and distribute the heat.

Gate discharge diodes were connected in parallel to the gate resistors, to empty the gate faster, and thus decrease the turn-off time. Two LEDs are connected in parallel to the load to give a visual indicator of the polarity and the (effective) potential of the voltage applied to the load.

The traces conducting the drive current to the motors were carefully placed, to keep them short and utilize most the available board area. Together with quite wide and thick (because of the solder applied) traces we achieved low resistance between the transistor and the load.

It was difficult to find suitable connectors for the load and power supply, due to the high currents involved. Our solution was to use a nut and bolt for each connection.

Testing and abandonment of first H-bridge

The first H-bridge functioned well when tested with a light bulb as load or when driving an electric engine with slow changes in speed. It died multiple times, however, when the speed was changed rapidly. We first suspected that the power transistors had died, but after changing them, it was apparent that it was the HIP4081 driver that had stopped working. The test setup is shown in figure 1.15.

We believe it was large transients across the motor that killed the driver. The HIP4081 driver is in fact connected directly to the motor terminals as it is using the switching frequency of the PWM-signal to generate the high voltage needed to turn «on» the top N-channel MOSFETs.

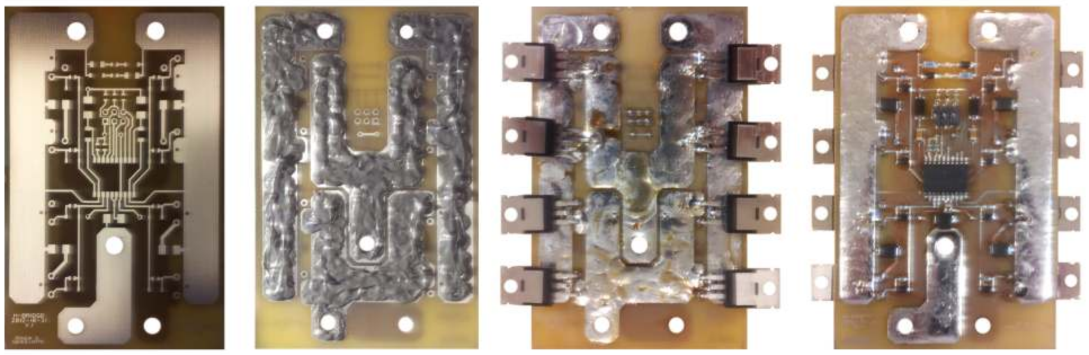
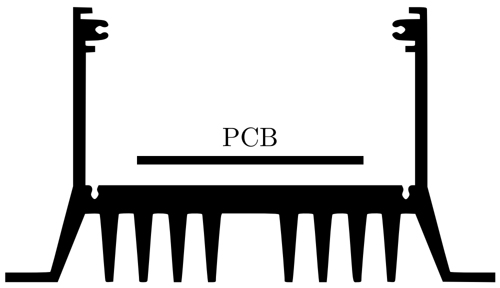
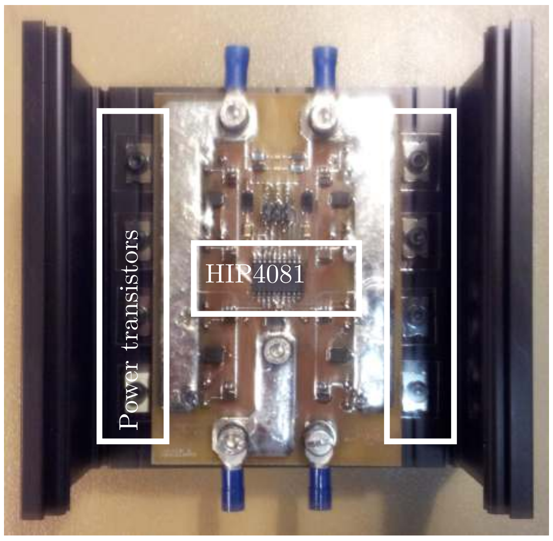


Figure 1.16: First H-bridge with solder paste and after soldering.



(a) Side view of the heatsink SK 61/100 SA from Fischer Elektronik.



(b) First H-bridge mounted in heatsink.

Figure 1.17: Heatsink for the first H-bridge.

1.3.2 Second attempt on H-bridge construction

Due to the low robustness of the first H-bridge, we set out to build a new version with our own controller and drive-stage. In this way we would have total control over the whole system, and we could distance the driver from the high voltage spikes from the motor.

The second motor control system incorporates one steering controller, one H-bridge controller, two H-bridge drive-stages and two H-bridges. The steering- and H-bridge controller, along with two H-bridge drive-stages are combined into one (H-bridge driver) PCB. Each of the H-bridges uses their own separate PCB. Three PCBs are thus needed for a full motor control system.

Motor control system stack

To interconnect the full motor control system, two solutions were considered; *a*) connecting the PCBs to a backplane; or *b*) stacking the PCBs. Even though a backplane construction gives easier access to each card, stacking the PCBs was chosen. The reason being the requirement of bulky and hard to source connectors needed to conduct the large currents through a backplane connection. This was chosen because the large currents involved would have required bulky and hard to source connectors for a backplane connection. The stack lets the current be distributed by bolts.

The stack also allows for multiple H-bridges to be connected in parallel. This makes it theoretical possible to control even higher currents than one H-bridge could handle. The H-bridge drive-stage limits how many transistors that can be driven at once, however.

The motor control system stack (with the prototype driver) is shown in figure 1.23b and (with finished driver) in figure 1.29.

An assembled stack is occasionally referred to as a «control unit».

Second steering- and H-bridge controller

The new steering controller is based on the previous digital dual channel steering controller. Since we used a multipurpose MCU, it can also be programmed to behave like an H-bridge controller.

Originally the same MCU breakout board containing an ATxmega128A1 was used, as shown in figure 1.23a. Later on, a custom PCB was designed (described in section 1.3.2) containing a smaller and newer version of the ATxmega128A1, namely the ATxmega32A4U (cost; ~3 USD each at a quantity of 100). In the new version (marked with a trailing «U» character), the manufacturer has corrected many of the silicon errors previously present and added Universal Serial Bus (USB)-functionality. These MCUs were chosen because of their motor driving capabilities, as well as the Input Capture interrupts described in section 1.3.1. They have specialized hardware for controlling half-bridges, as an extension of their PWM functionality. The extension is called Advanced Waveform eXtension (AWeX) and gives the ability to insert a dead-time and invert one of the outputs, so the top and bottom transistors never are turned «on» at the same time. Shoot-through conditions should not be possible if this is set up correctly.

The firmware for the MCU is available at `https://github.com/epsiro/ASV_H-bridge_controller` and is listed in appendix G. Git[7] is used for version control of the software. The firmware reads the pulse-widths, finds the stick position, calculates the appropriate motor thrust and creates eight (high and low side) PWM-signals with dead time. It will skip the first commands (pulse-widths) received from the RC-receiver, to wait for the RC-transmitter and -receiver to initialize. If the connection with the RC system is lost, a timeout will automatically stop the motors within a short amount of time.

[7]Git is a free (as in free speech) distributed revision control system

A block schematic of the program flow is shown in figure 1.18. Some of the blocks are outlined with pseudo-code in the following listings.

Listing 1.1: normalize_stick_position

```
center pulse-width around 1.5 ms
insert dead band
scale to -127..127
round up 125..127 to 127
```

Listing 1.2: combo_sticks

```
if (left_stick == 0 && \
        right_stick != 0)
    turn on the spot
    motor_thrust_left  =  right_stick;
    motor_thrust_right = -right_stick;
else
    normal drive
    motor_thrust_left  = left_stick +
        right_stick;
    motor_thrust_right = left_stick -
        right_stick;

remove overflows

drive left motor through a low pass
    filter (drive_motor_lp)
drive right motor through a low pass
    filter (drive_motor_lp)
```

Listing 1.3: drive_motor_lp

```
calculate moving average
drive motor (drive_motor)
```

Listing 1.4: drive_motor

```
scale the pwm-value
if (motor_thrust > 0)
    drive forward // set timer reg.
else
    drive backward // set timer reg.
```

The firmware can also receive steering commands over UART. This can be used to communicate with another control system, e.g. an autopilot. The steering commands consist of one byte, where the most significant bits (MSBs) indicate which motor to control, the second MSB indicates the direction and the remaining 6 bits sets the speed (in 64 steps), as shown in table 1.2. Each motor will run at the given speed until a new command is received or a timeout is encountered (after missing 20 UART commands). The timeout will stop the motors. This simple binary format was found to be sufficient, and was chosen for simplicity over a more complicated ASCII or multi-byte binary format.

Second H-bridge drive modes

Static operation of an H-bridge is described in section «H-bridge operation» on page 8. The following section will describe how the second H-bridge controller adjusts the speed and direction of the motors.

As described earlier, the H-bridge are controlled by multiple PWM signals, where their pulse-width set the speed of the motor. These PWM signals are generated by the MCU on the H-bridge driver PCB and amplified by the H-bridge drive-stage's push-pull circuit. The signals generated by the H-bridge driver (connected to one of the H-bridges) is shown in figure 1.19. By comparing this figure with table 1.5b on page 8, we can see which H-bridge states we are using. We are alternating between the state «motor runs cw» (S1) and «motor brakes» (S5), when we drive forward, and «motor runs ccw» (S2) and «motor brakes» (S5), when we drive backward. The reason why we «brake» in between each cycle is described in the following paragraph.

Since our load is inductive, energy will be present even after the switches are turned «off» and the current have stopped. To release this energy the inductive load drives a current back (back-EMF). We need to provide a path for this current to flow. If we do not provide this path, the inductive load will increase the voltage across itself, as the energy it harvests has nowhere else to go. This will result in a high voltage that is harmful to the switching transistors. To provide this path, we turn both the top transistors «on» so the energy can escape back to the power supply. Large capacitors, which can consume this sudden backward current, are connected in parallel with the supply rail to protect the power supply.

By turning both top transistors «on» we risk entering a shoot-through state, since one of the bottom transistors was already «on» when the motor was actively driven. To prevent two transistors on one side from being «on» at the same time, the MCU inserts a time delay after the bottom transistors have been turned «off», and before the top transistor is turned «on». This delay is called a dead-time. The dead-time

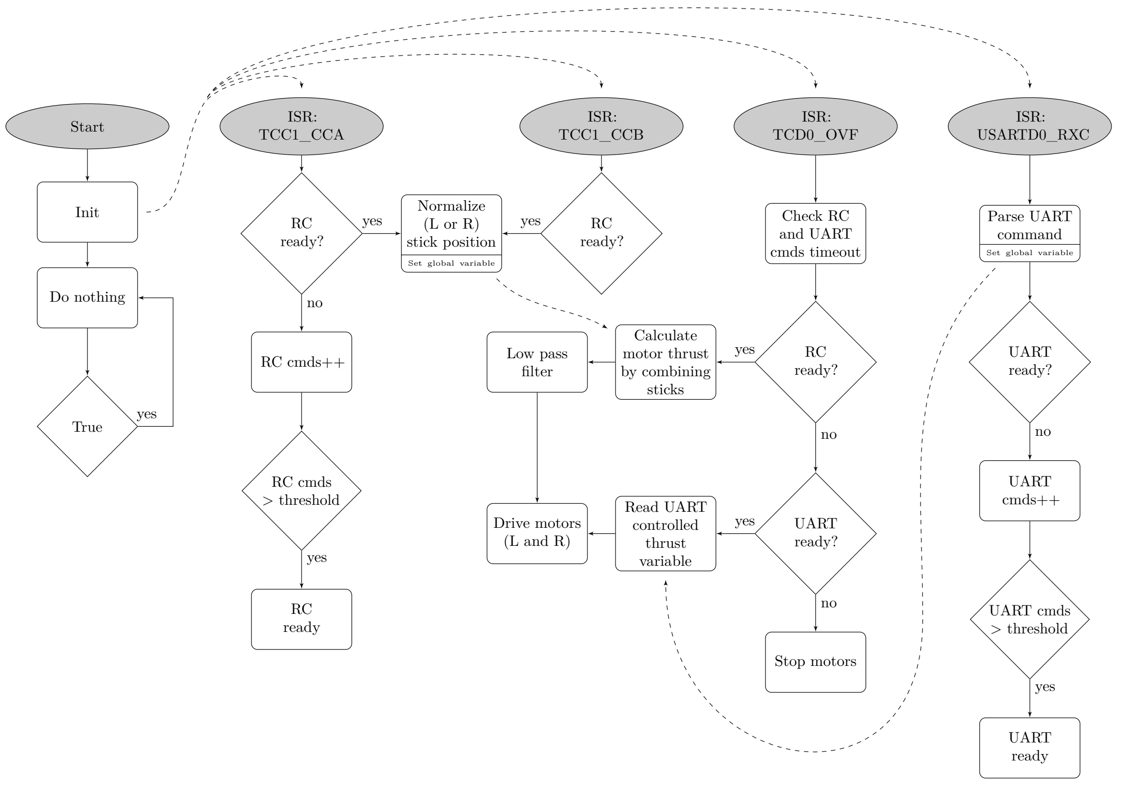


Figure 1.18: Block schematic over the H-bridge controller program flow. The ISR TCC1_CCA and TCC1_CCB is triggered every time a RC-control pulse is received. The ISR TCD0_OVF is triggered every time a timer overflows (100 Hz). The ISR USARTD0_RXC is triggered every time a byte is received over UART.

Table 1.2: Example of selected UART steering commands.

	Motor	Direction	Speed	Behaviour
0b	0	0	000000	Left motor 0 % forward (stop)
0b	0	1	111111	Left motor 100 % backward
0b	1	0	011111	Right motor ~49 % forward
0b	1	1	001111	Right motor ~24 % backward

is long enough to last through the turn-on and turn-off time of both transistors, along with an added safety margin. We insert a dead-time of 30 μs.

The flyback diodes provide the necessary path back to the power supply during the dead-time (when both transistors are «off»). They are however not capable of handling the back-current for a long time, as they would overheat[8].

Second H-bridge drive-stage

To minimize the time it takes to turn the MOSFETs in the H-bridge «on» and «off», their gates needs to be quickly charged and discharged. This is accomplished by using a BJT push-pull stage to drive the MOSFETs, as shown in figure 1.21. The push-pull drive-stage is again driven by a singe BJT, to amplify the drive signal from the MCU, as the MCU only can source a few milliamperes itself. This inverts the drive signal which must be compensated for in the MCU, either in software or by inverting the IO-pins.

The push-pull drive-stage was prototyped on a wooden plank to drive a half-bridge using the IRF1404z power transistors, as seen in figure 1.20. A half-bridge is a set of just one top and bottom transistor, simply half of the H-bridge, with the other side of the load always connected to ground. This setup could successfully change the brightness of a lamp or the speed of an electric engine quickly without dying. Since this was only a half-bridge, it could not change the direction of the current.

Voltage multiplier

To turn «on» the high N-channel MOSFETs in the H-bridge, a high voltage source is needed. A voltage multiplier is used to create this voltage.

It consists of an oscillator (astable multivibrator) and a diode/capacitor-ladder. From figure 1.22 you can see the 12 V input flow through diode D1 and charge capacitor C3. When the oscillator goes high, the voltage below C3 goes up to 12 V. Since C3 still has a voltage of almost 12 V across itself, the voltage after D1 is lifted up to almost 24 V. This high voltage can now go through the second diode D2, and charge capacitor C2.

We now get almost twice the input voltage out, minus the two diode drops. Although the effective voltage can be much lower, as the voltage doubler only can source a limited current. Multiple ladders can be cascaded to further multiply the voltage, however the current sourcing capability will be even lower.

Second H-bridge driver prototype

After the push-pull drive-stage was shown to successfully drive a half-bridge and the voltage multiplier was designed, a complete driver for two H-bridges was prototyped on a veroboard. The prototype is shown in figure 1.23a. A full motor control system using this prototype, is shown in figure 1.23b.

[8]Since their constant voltage drop multiplied with the current results in a high power dissipation

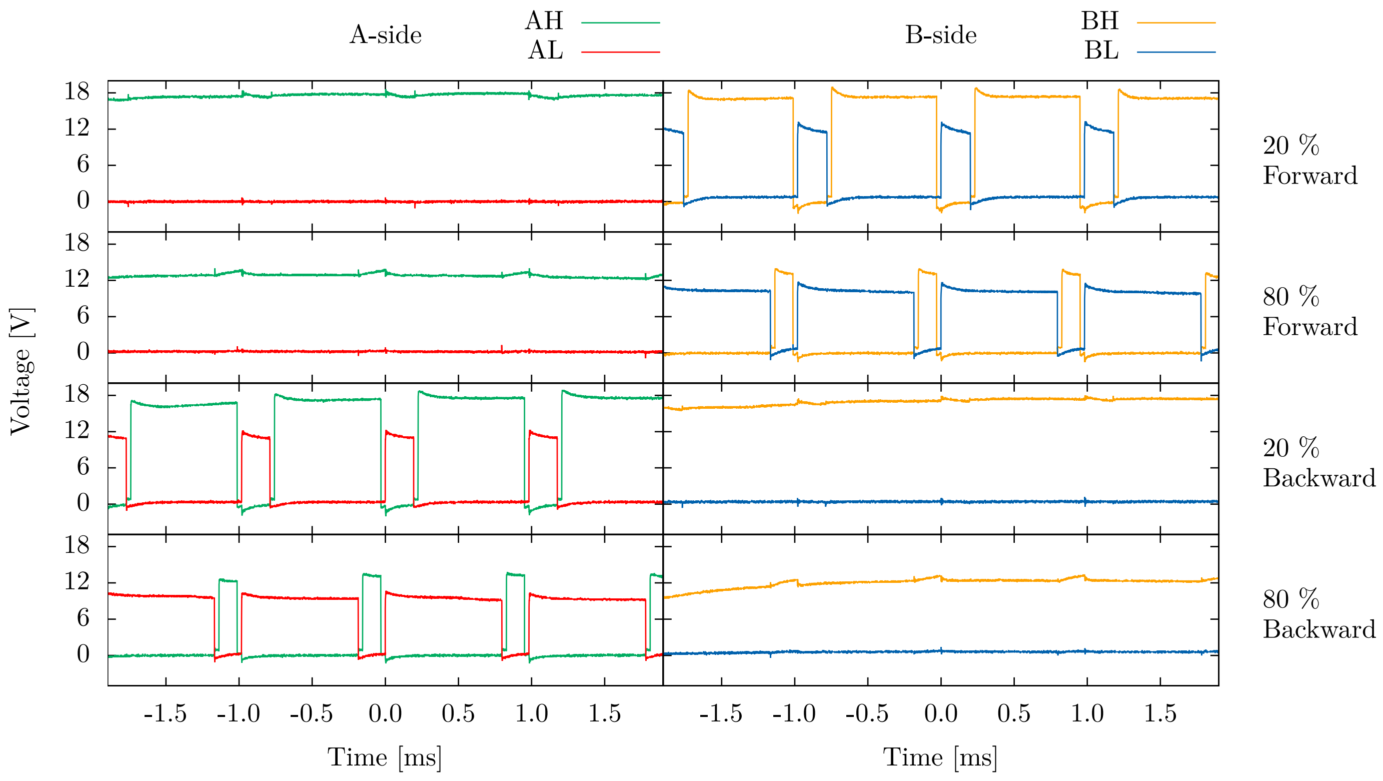


Figure 1.19: Drive signals from the H-bridge driver to the four legs of one of the H-bridges. AH, AL, BH, BL denotes the drive signal for the A- and B-side high (top) and low leg. The two top rows shows the signals used to drive the motor forward at two different speeds. Likewise, the two bottom rows shows backward driving. Forward and backward-driving is clearly very similar, with just the A- and B-side switched.

Second selection of power transistors

Since the transistors and the heatsink of the first H-bridge never got noticeable warm while driving the electric engine, I decided to replace them with surface mounted transistors and use the PCB itself as the heatsink. This would make manufacturing considerable easier, and lower the cost as the external heatsink is no longer needed.

I chose PSMN2R6-40YS from NXP, because it *a*) has a low $R_{ds_{on}}$ of only 2.8 mΩ (at $V_{GS} = 10\,\mathrm{V}$, $I_D = 25\,\mathrm{A}$ and T_j[9] $= 25\,^\circ$ C); *b*) is built in a package with a high power dissipation (~130 W at 25 ° C); *c*) has a low gate charge; *d*) can withstand a continuous current of 100 A (peak ~650 A); *e*) is inexpensive (~0.9 USD each at a quantity of 100); *f*) has a drain-source voltage of 40 V; and *g*) has embedded flyback diodes. The package used by this transistor is the Loss-Free Package (LFPAK) (figure 1.24), a type of Power-SO8. LFPAK eliminates the bonding wire commonly used in Power-SO8 packages, by soldering a copper clip directly to the silicon die, to connect the source and the gate with very low resistance. This allegedly results in superior electrical and thermal performance as well as higher reliability [13], according to the manufacturer. The bottom of the silicon die is soldered to a drain tab to provide a low thermal resistance path down to the PCB.

[9] T_j is the junction temperature

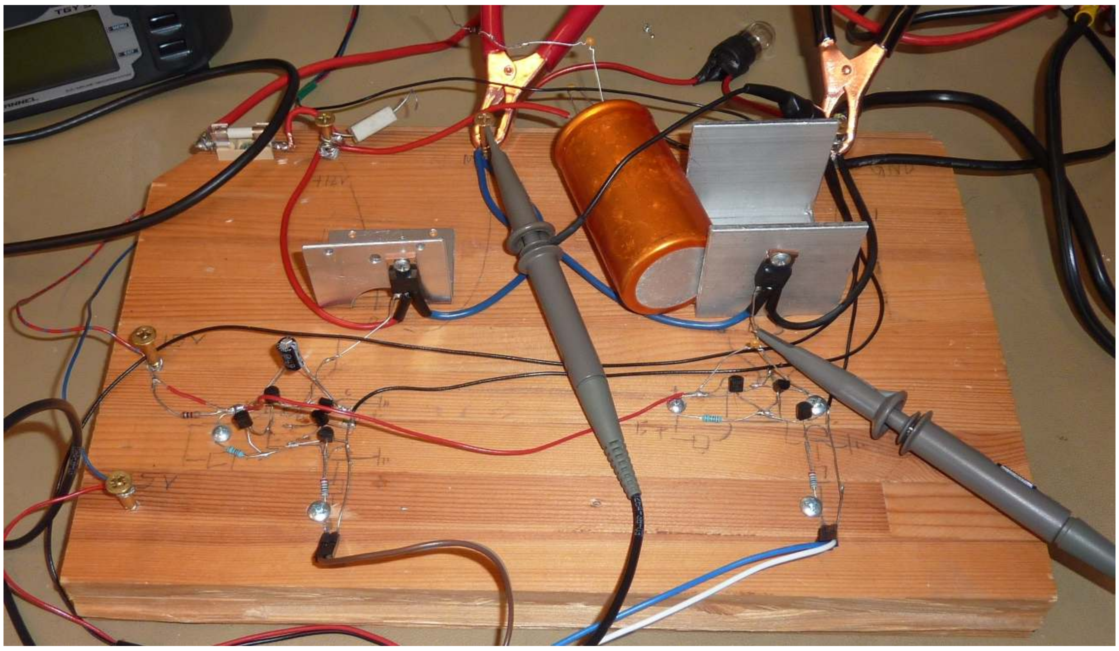


Figure 1.20: Photograph of the half-bridge prototype.

Table 1.3: Power transistors families compatible with the universal Power-SO8 footprint.

Manufacturer	Device Family
NXP	LFPAK (SOT669/SOT1023)
Infineon	PG-TDSON-8
Fairchild	Power 56
Vishay	PowerPAK SO-8
NEC	8-pin HVSON
ON Semi	SO-8 FL
STM	PowerFLAT (6x5)
Renesas	LFPAK

I have used the recommended universal Power-SO8 and LFPAK footprint which according to [13] allows all the device families listed in table 1.3 to be soldered. If another transistor is found to be better suited, it should thus be possible to solder it to the same PCB. The footprint is shown in figure 1.25.

Some of the later H-bridge PCBs has used the 60 V drain-source version (PSMN7R0-60YS), since PSMN2R6-40YS was temporarily out of stock. PSMN7R0-60YS has a $R_{ds_{on}}$ of 6.4 mΩ (at $V_{GS} = 10\,\text{V}$, $I_D = 15\,\text{A}$ and $T_j = 25\,°\,\text{C}$).

Second H-bridge design and production

After the push-pull drive-stage was successfully prototyped with a half-bridge, I wanted a bare H-bridge PCB to continue development. I designed the PCB shown in figure 1.27, which only contains the power transistors and no driver. This was done to maximize the copper usage of the PCB, as the copper planes serve as the heatsink for the power transistors. I used four power transistor in each leg of the H-bridge, to distribute the heat and current.

Four transistors in parallel gives an effective $R_{ds_{on}}$ of only 0.9 mΩ (even with a high junction temperature)[10] and with a current of 40 A equation (1.1) yields a static power dissipation of about 1.5 W.

As suggested by the application note [14], I placed a lot of copper around each transistor, maximized the distance between them, and used multiple layers with thermal vias to transfer

[10] $R_{ds_{on}}$ at $V_{GS} = 10\,\text{V}$, $I_D = 25\,\text{A}$ and $T_j = 100\,°\,\text{C}$. Although a higher current of 40 A does not influence the $R_{ds_{on}}$, when the V_{GS} is as high as 10 V.

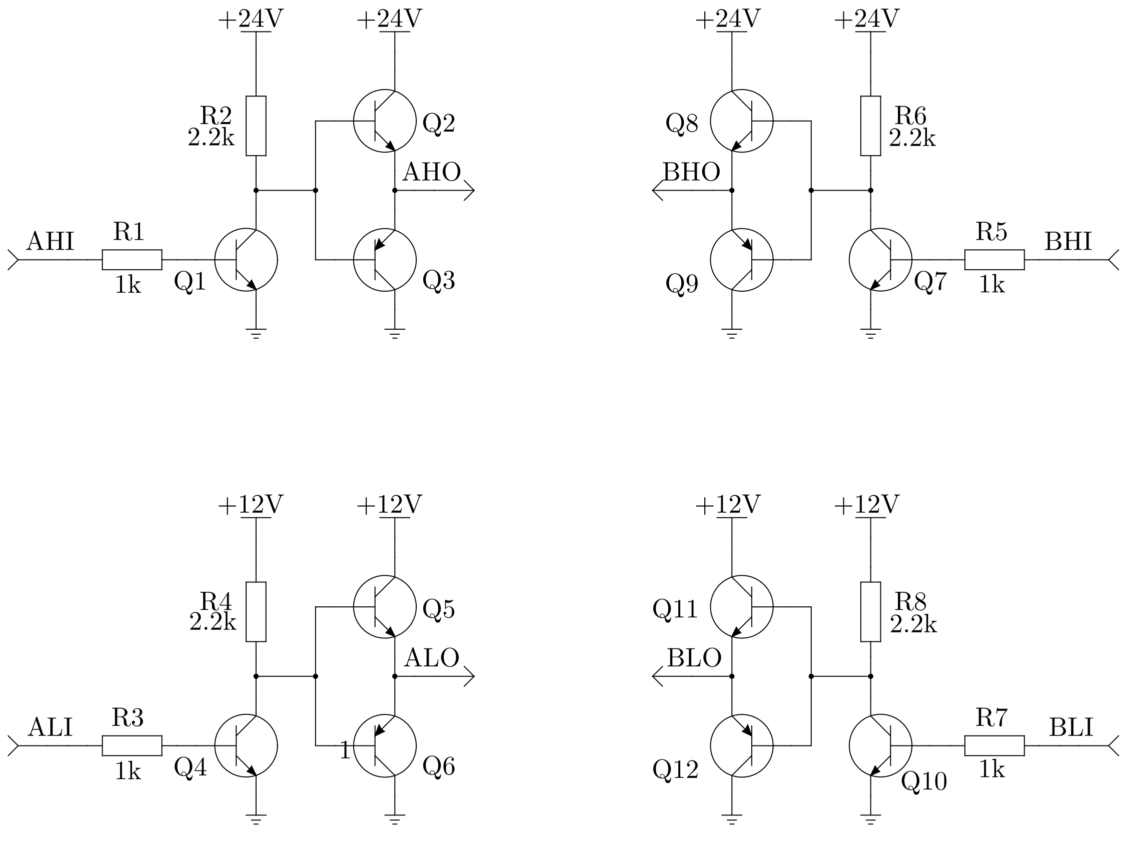


Figure 1.21: Schematic of the push-pull stage of the H-bridge drive-stage.

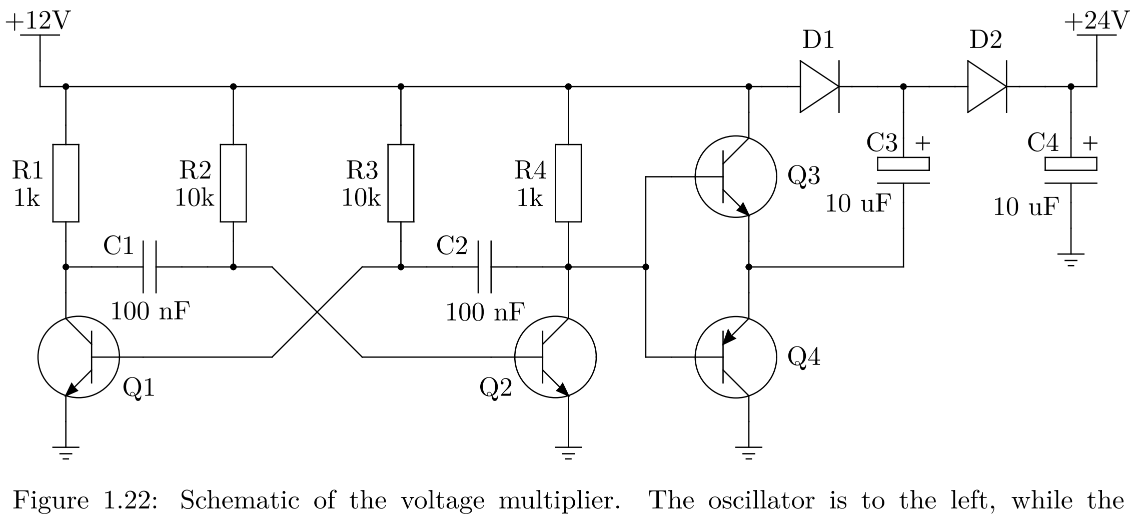
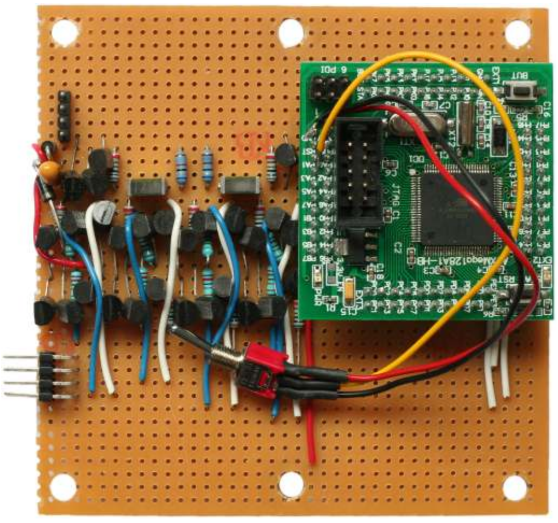


Figure 1.22: Schematic of the voltage multiplier. The oscillator is to the left, while the capacitor/diode ladder is to the right.


(a) Prototype of the full H-bridge driver for two H-bridges. With the steering and H-bridge controller to the right (green board), and the drive-stage to the left. The voltage multiplier is located above the drive-stage.

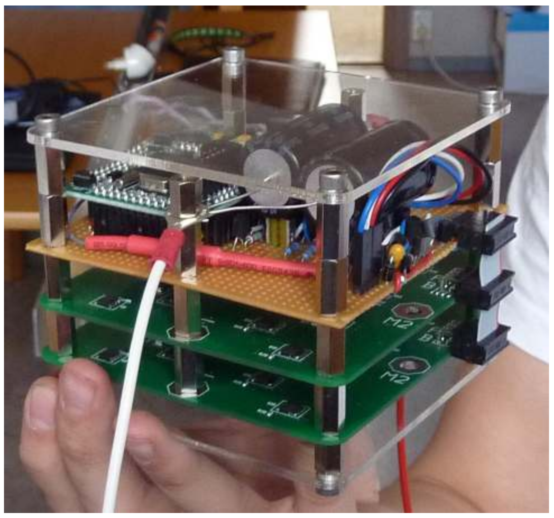

(b) Prototype of the full motor control system stack (control unit). With H-bridge driver for two H-bridges (top), and H-bridge for left motor (middle) and right motor (bottom). The RC-receiver is located in the back of the top PCB, and not visible. Laser-cut acrylic sheets form a very simple cover.

Figure 1.23: Prototype of H-bridge driver and full motor control system.

heat away. We used a 2-layer standard PCB with a copper thickness of 35 μm. Based on the equations described in section 1.2.3 and the use of copper planes that are nearly square, the resistance for the drive current will be low even with 35 μm copper. The current is also distributed on two layers and between multiple transistors, so the power dissipation is low enough for the copper sheets to not overheat during normal use.

As done in the first attempt, additional flyback diodes and indicator LEDs are included. Series resistors are connected to each gate of the power transistors, to limit the gate current (and turn-on and turn-off times) and dampen ringing between the gate's capacitance and wire's inductance. A value of 220 Ω was initially chosen, but this can be decreased. The gate discharge diodes were not included, because they were deemed unnecessary with the new drive-stage. There is room on the PCB for a transient-voltage-suppression (TVS) diode.

The gates of the power transistors are connected to a 8-pin connector. Each leg in the H-bridge (four transistors) can be connected to two pins on this connector. By soldering 0 Ω resistors, the PCB can be configured to use the top (marked A) or bottom (marked B) side of the connector[11]. By doing so, the same PCB layout can be used to create two H-bridges, which can be controlled separately. This is needed since we wish to control two electric engines.

An assembled H-bridge is shown in figure 1.26a.

Second H-bridge driver design and production

After the second H-bridge driver had been prototyped on a veroboard, I designed a PCB containing the same circuit. The PCB consists of an ATxmega32A4U MCU, a voltage multiplier, two H-bridge drive-stages and connectors.

[11]Not to be confused with the A- and B-side of the H-bridge

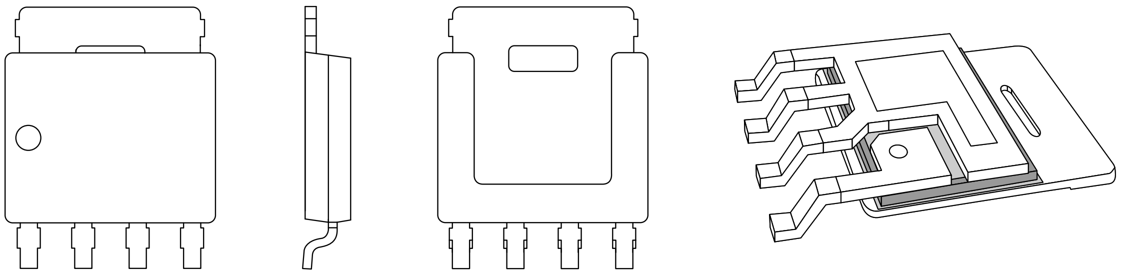


Figure 1.24: Illustrations of the LFPAK package. From the left: Front view, side view, back view, internal view.

The ATxmega32A4U is a smaller and cheaper version of the previously used ATxmega128A1, and the same code can run on both (although they have different peripherals available). The bare PCB for the H-bridge driver is shown in figure 1.28, and a assembled board is shown in figure 1.26b.

1.3.3 Light version of the final H-bridge construction

Since the second motor control system worked very well, but used a lot of space, a light version was constructed as well. This can be useful for operating smaller motors. This unit incorporates two H-bridges with two drivers on one PCB, measuring only 5 × 10 cm. It is capable of driving loads of up to 8 A. The PCB is shown in figure 1.30.

There was an error with the pinout of some of the transistors, so the board is currently not functional.

1.4 Testing and results

The complete motor control system, using the prototype of the H-bridge driver, has been tested at Lysaker, Norway and on lake Římov, Czech Republic. A temperature test has also been performed.

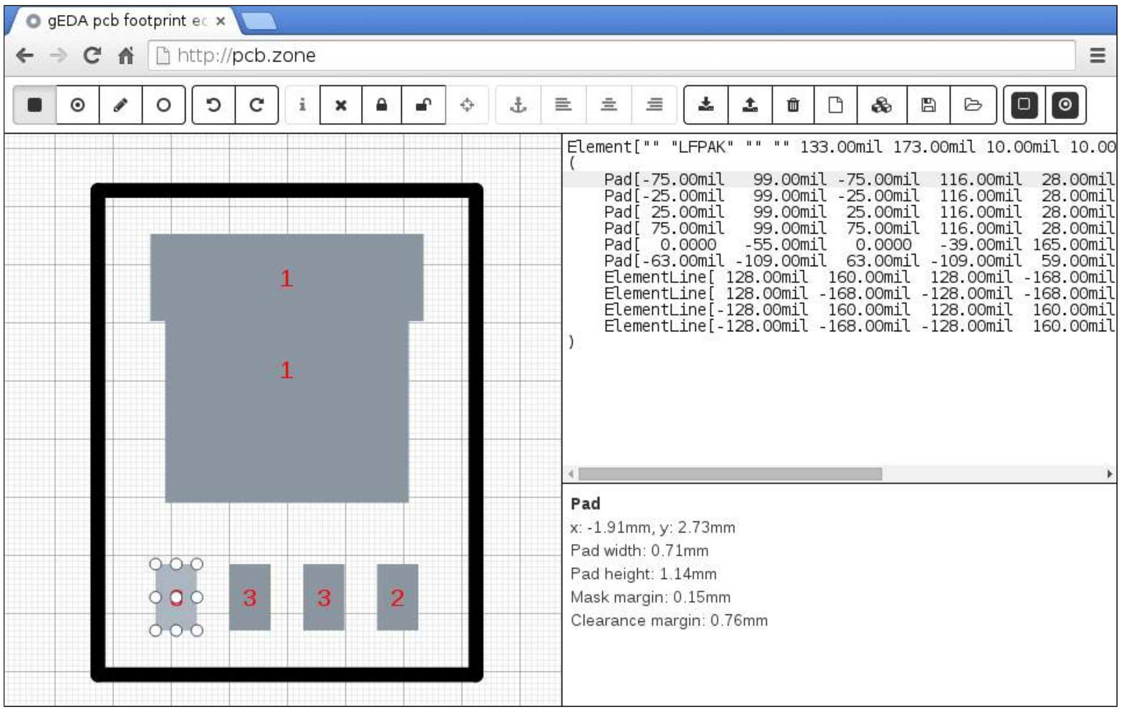
1.4.1 Test at Lysaker

A complete motor control system (shown in figure 1.23b), with two of the second version H-bridges and the prototype of the H-bridge driver, was tested May 2th, 2013 at Lysaker, Norway. The control unit (assembled motor control system) was placed in an 8 feet boat, together with two electric outboard engines and a lead-acid battery. The boat is shown in figure 1.31 and 1.32b, and the control unit in figure 1.32a. The engine's steering handles were mounted in a fixed position using wooden bars, to keep them from rotating.

Both steering mechanisms were tested. The most primitive steering mechanism (with each stick individually controlling one motor) was very difficult to use. The combination steering mechanism, however, (with one stick controlling the thrust and the other controlling the direction) was surprisingly easy to master. The primitive steering mechanism was thus abandoned in favor of the combination steering.

After testing the system with a person on board, the boat was released without a driver, although with a fishing line for security. The boat was successfully driven in multiple patterns by an operator at shore, and showed great maneuverability. The speed of the boat was also rapidly changed from stand-still to full-speed, and halted again, without damaging the control unit.

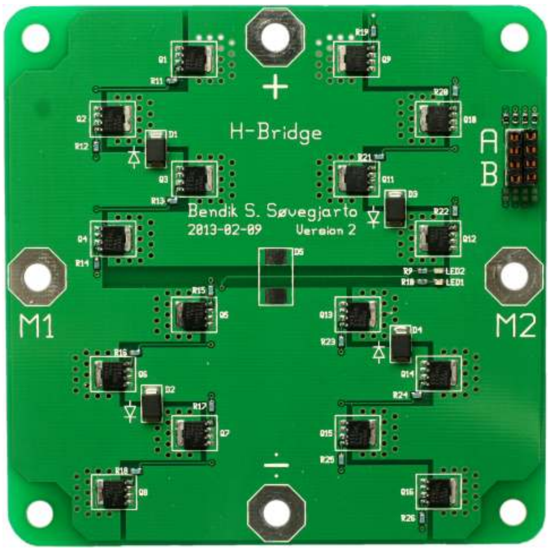
[12]Fish Ecology Unit of the Department of Fish and Zooplankton Ecology of the IH BC CAS. `http://www.fishecu.cz/`



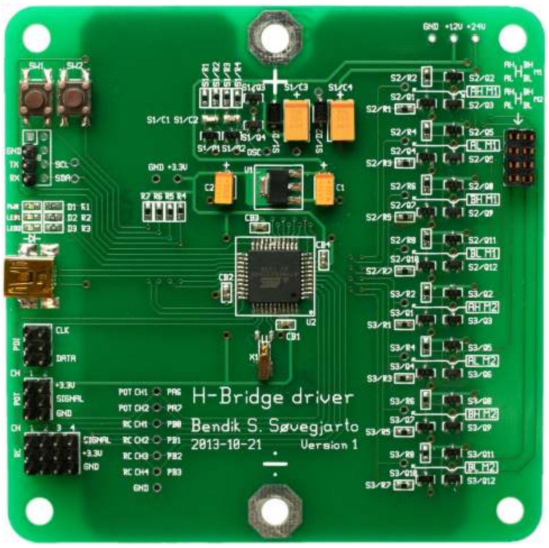
```
Element["" "LFPAK" "" "" 133.00mil 173.00mil 10.00mil 10.00
(
    Pad[-75.00mil   99.00mil -75.00mil  116.00mil  28.00mil
    Pad[-25.00mil   99.00mil -25.00mil  116.00mil  28.00mil
    Pad[ 25.00mil   99.00mil  25.00mil  116.00mil  28.00mil
    Pad[ 75.00mil   99.00mil  75.00mil  116.00mil  28.00mil
    Pad[  0.0000   -55.00mil   0.0000   -39.00mil 165.00mil
    Pad[-63.00mil -109.00mil  63.00mil -109.00mil  59.00mil
    ElementLine[ 128.00mil  160.00mil  128.00mil -168.00mil
    ElementLine[ 128.00mil -168.00mil -128.00mil -168.00mil
    ElementLine[-128.00mil  160.00mil  128.00mil  160.00mil
    ElementLine[-128.00mil -168.00mil -128.00mil  160.00mil
)
```

Pad
x: -1.91mm, y: 2.73mm
Pad width: 0.71mm
Pad height: 1.14mm
Mask margin: 0.15mm
Clearance margin: 0.76mm

Figure 1.25: Universal Power-SO8 footprint, although especially made for LFPAK. Displayed using my self-developed footprint editor. Code available at `https://github.com/epsiro/pcb-footprint-editor`. Demo available at `http://pcb.zone` in 2015.



(a) Assembled PCB of second H-bridge.



(b) Assembled PCB of second H-bridge driver.

Figure 1.26: Photographs of assembled PCBs for the second H-bridge and H-bridge driver.

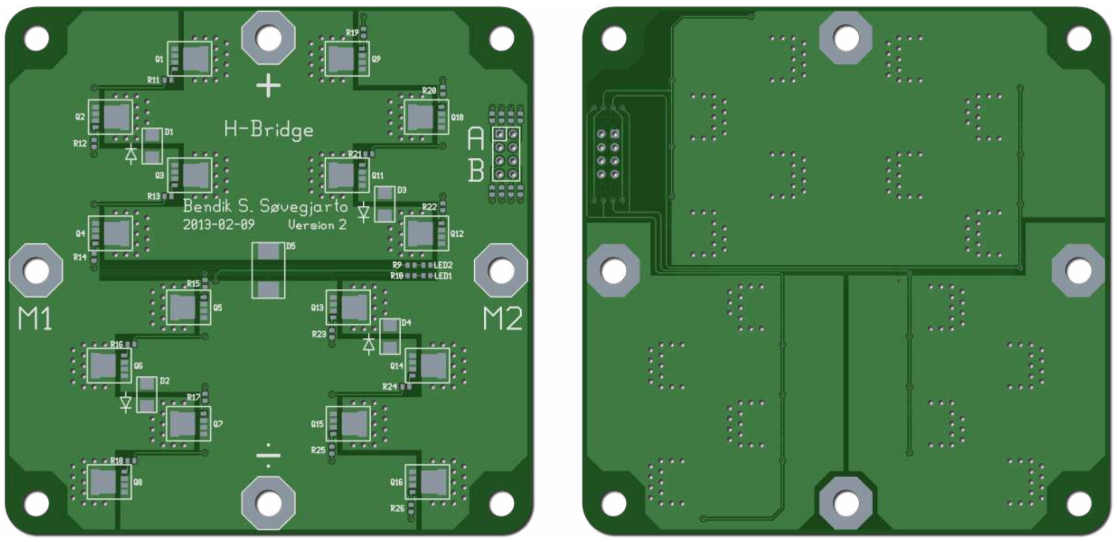


Figure 1.27: CGI of the PCB for the second H-bridge. Left image shows the top side, right image shows the bottom.

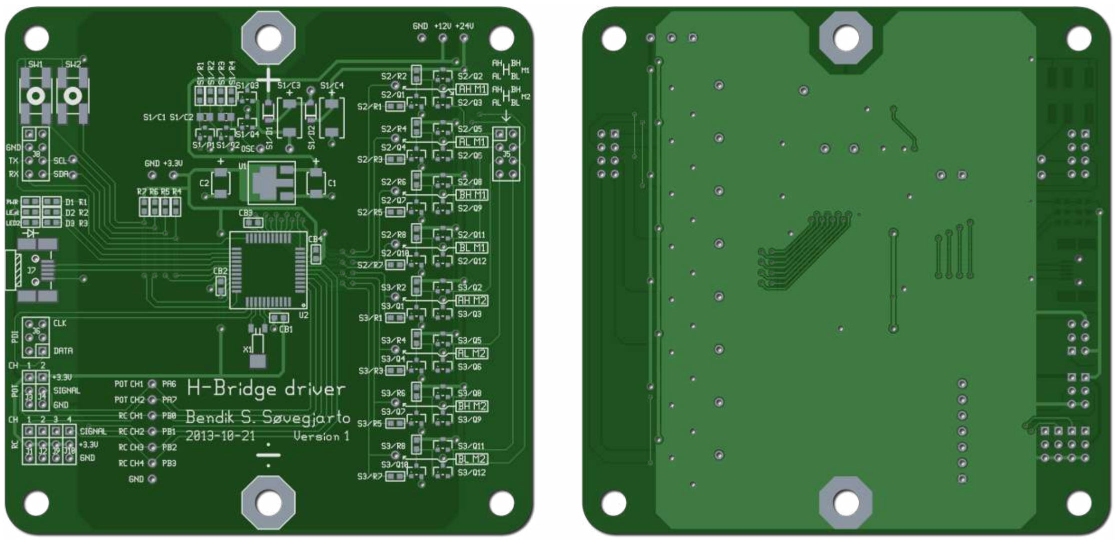


Figure 1.28: CGI of the PCB for the H-bridge driver. Left image shows the top side, right image shows the bottom.

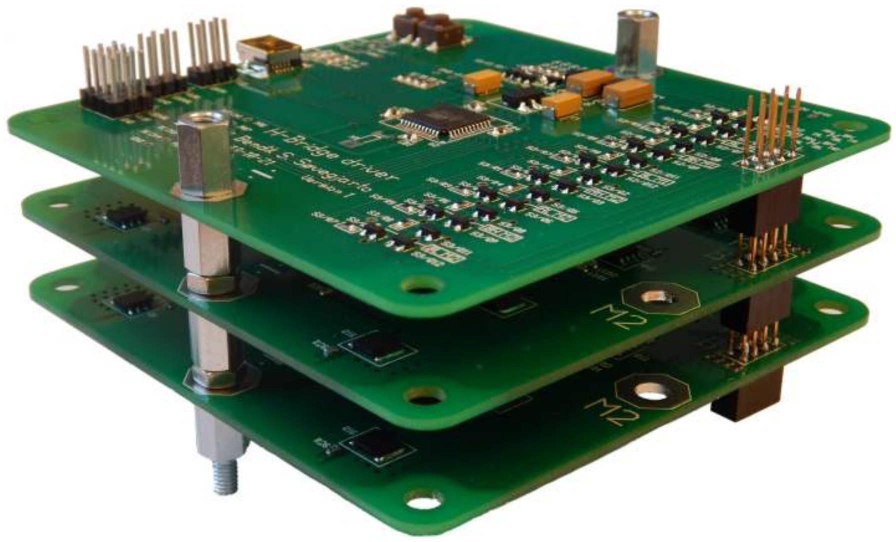
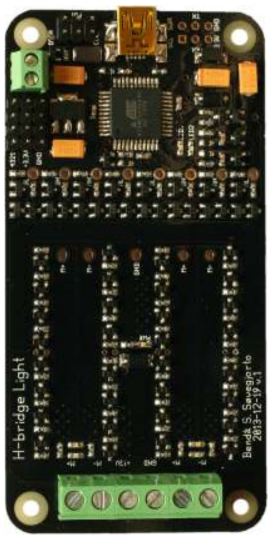
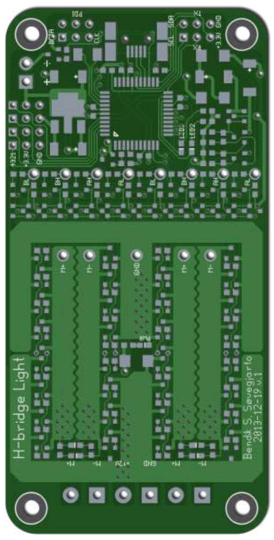


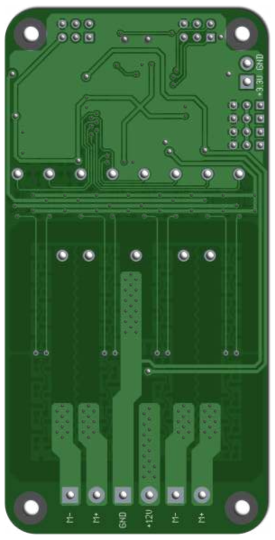
Figure 1.29: Photograph of finished motor control system stack (control unit).



(a) Photograph of assembled H-bridge light PCB.



(b) CGI of H-bridge light PCB top side.



(c) CGI of H-bridge light PCB bottom side.

Figure 1.30: Assembled and CGI of H-bridge light PCB.

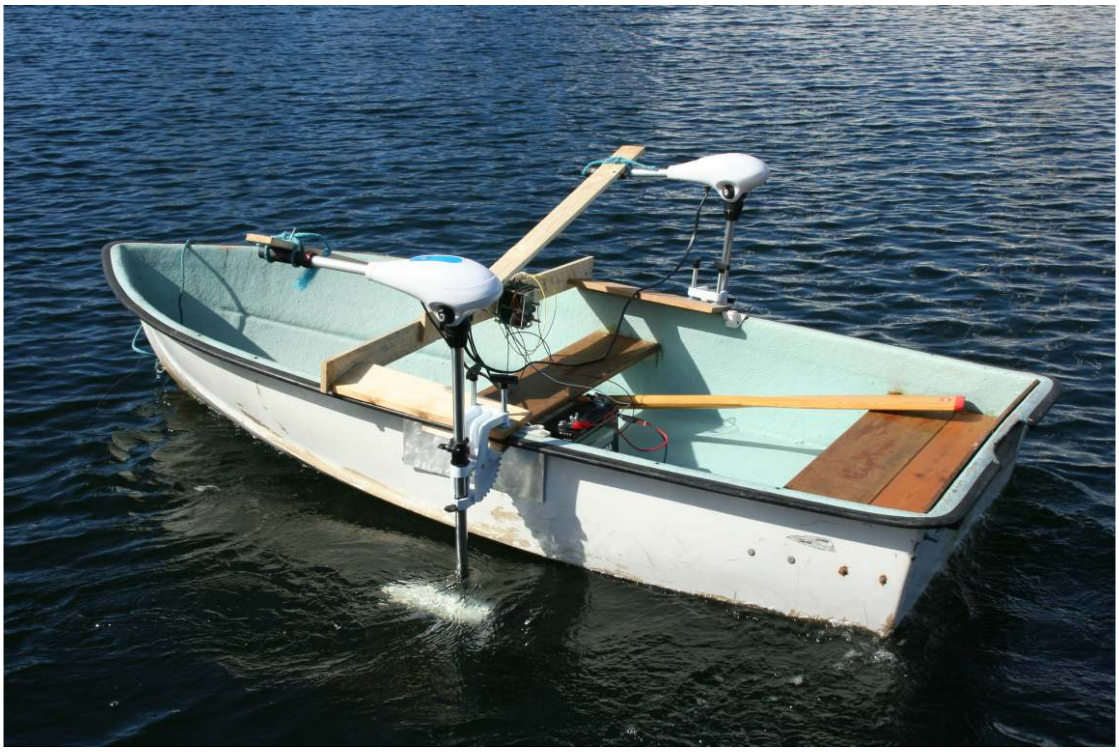
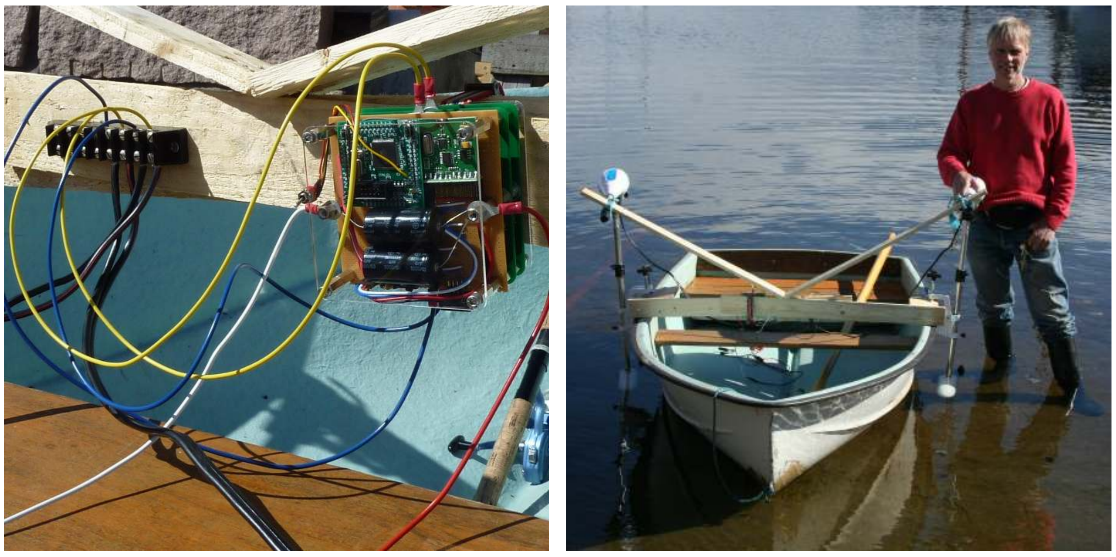


Figure 1.31: Photograph of boat, with electric engines and control unit, tested at Lysaker in May 2013.



(a) The control unit mounted in the boat.

(b) The boat during deployment, with dr. Helge Balk as scale.

Figure 1.32: Detailed image and image to scale, of the boat used at the Lysaker test.

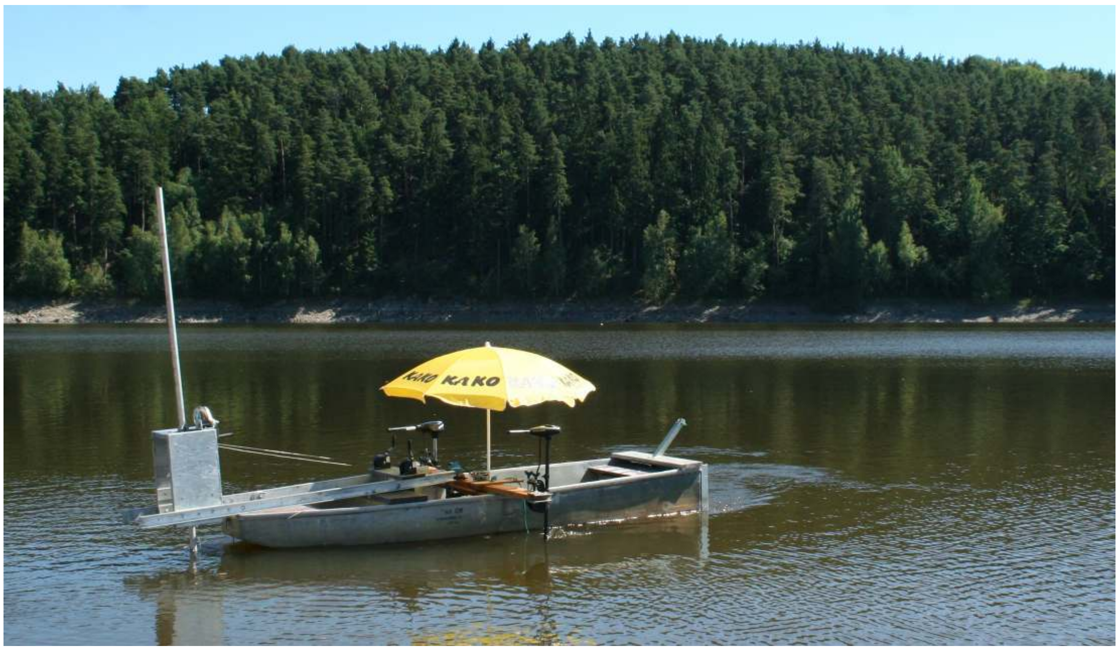
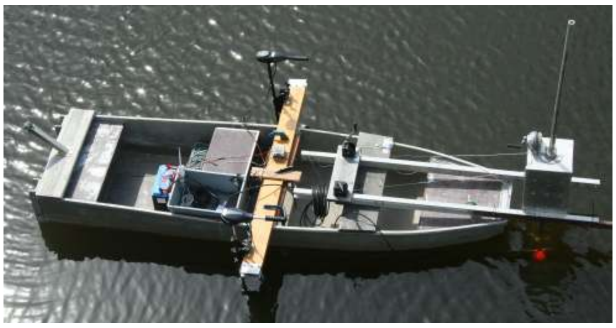
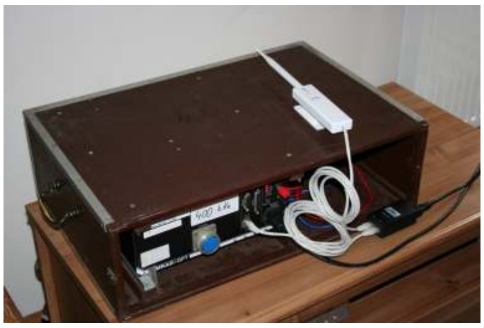


Figure 1.33: Photograph of aluminium boat used on lake Římov.



(a) Photograph of boat with a Simrad EK60 GPT.



(b) Simrad EK60 GPT directly connected to a Ubiquity PicoStation M2 (through a POE-injector).

Figure 1.34: Simrad EK60 GPT with boat and PicoStation.

1.4.2 Test on Římov

The same control system as tested at Lysaker was tested on lake Římov in August 2013. A suitable vessel was provided, along with two (different) electric outboard engines, by FishEcU[12]. The vessel was a 12 feet long aluminium boat, shown in figure 1.33.

The boat was successfully driven in different patterns, up to about 100 meters from the operator. It was also tested at full speed, and used for longer periods of time (hours).

Some of the journeys were conducted with a Simrad EK60 400kHz General Purpose Transceiver (GPT), as shown in figure 1.34a. The data from the echosounder was transferred wireless to the shore using a pair of Ubiquity PicoStation M2. To view the echogram live we first attempted to connect by remote desktop to a computer located in the boat. The result was not usable, as the refresh rate was very low. I therefore configured the radio link and the GPT to forward the User Datagram Protocol (UDP) packages from the GPT directly through the link and to a computer located at the shore. The setup with the echosounder and the link is shown in figure 1.34b. This worked very well, and gave the same usability as sitting in the boat. The data from these test surveys were viewed and recorded to the computer at the shore. We now had a working remote controlled survey vessel, as shown in figure 1.35.

I wrote a C-application to parse the recordings, along with a web-based (JavaScript/-HTML/CSS) echogram viewer[13]. The parser was later ported to JavaScript as well. One of the recordings taken with the remote controlled survey vessel is shown in figure 1.36.

The boat was also remotely controlled to move a hydroacoustic standard target at a fixed depth along a straight path. This was done to inspect oscillating phenomena occurring during horizontal beaming. This worked successfully, although the larger boat was more challenging to maneuver than the boat used at Lysaker.

The tests conducted on lake Římov showed that the control system could successfully be used as a remote work platform, carrying different payloads. It was also capable of working with the hardware available at hand.

[13]Demo available at `http://opensonar.net` in 2015.

1.4.3 Temperature test

A temperature test was performed to find out which parts of the H-bridge would get hot under high loads. The Biltema electric outboard engine was mounted in a big bucket with water as shown in figure 1.39a. An H-bridge and H-bridge driver was connected and the RC-transmitter was used to control the thrust. A self-made power resistor and a Hewlett-Packard (HP) 3455A voltmeter was used to measure the current, as well as a Kyoritsu 2300R fork current tester, as shown in figure 1.38. The power resistor was made using multiple turns of 2.3 $\frac{\Omega}{\mathrm{m}}$ resistance wire. Its resistance was measured to be 14.3 mΩ using an Agilent 4263B LCR meter. A Fluke Ti25 infrared (IR) camera was used to monitor the temperature.

The RC-transmitter was used to keep the thrust about ~90 %, which corresponded to a current draw of ~20 A. 90 % was chosen because this yields switching losses as well as a high «static» loss. The battery could unfortunately not keep up the current of 20 A for longer than five minutes. The figure 1.37a shows the current over the course of these five minutes, while figure 1.40 shows the temperature after one minute and after three and a half minutes.

The battery had enough energy for a new test with the motor running at a lower speed. The cable was securely reattached to the screw terminal, and the motor was run for another three minutes. The duty-cycle was adjusted with the RC-transmitter to keep a current draw of 5 A. The figure 1.37b shows the current over the course of these three minutes, while figure 1.41 shows thermal images of the whole H-bridge and a close up of one leg.

The results of the temperature test is discussed in section 1.5.3.

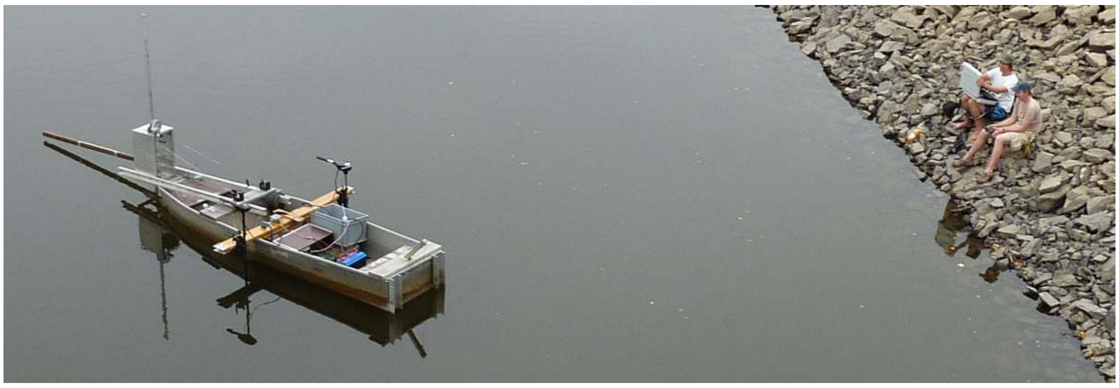
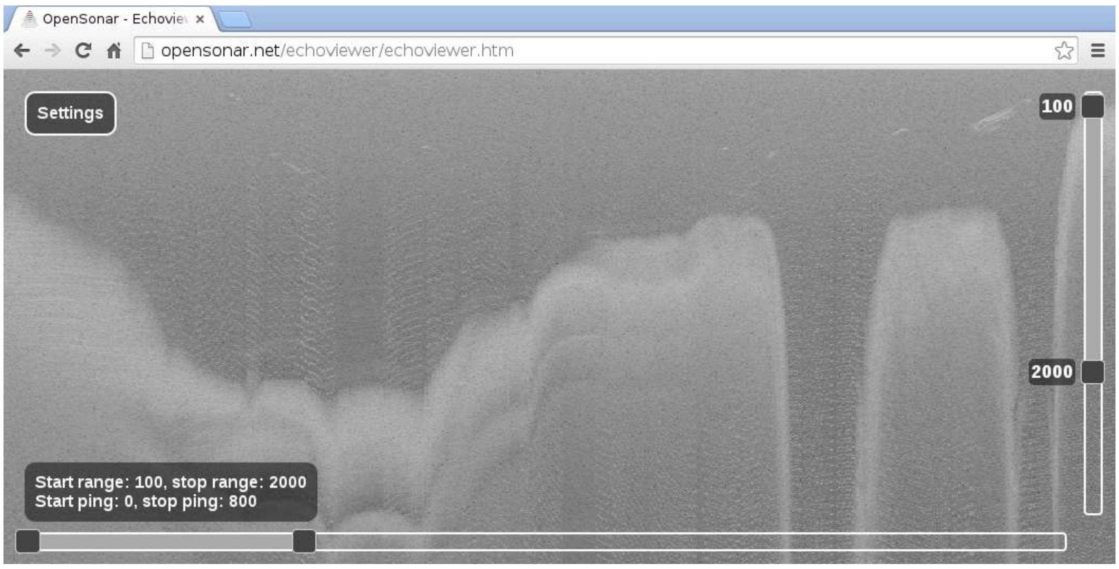
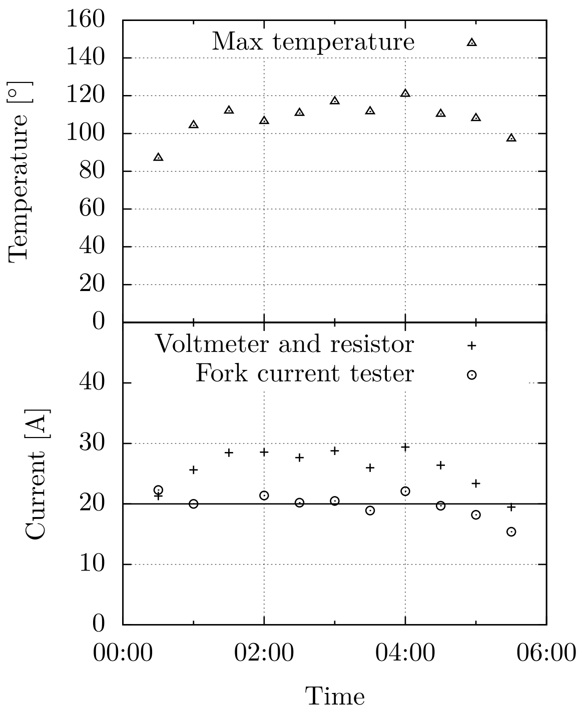


Figure 1.35: Remote controlled survey vessel supervised by operators at the shore.

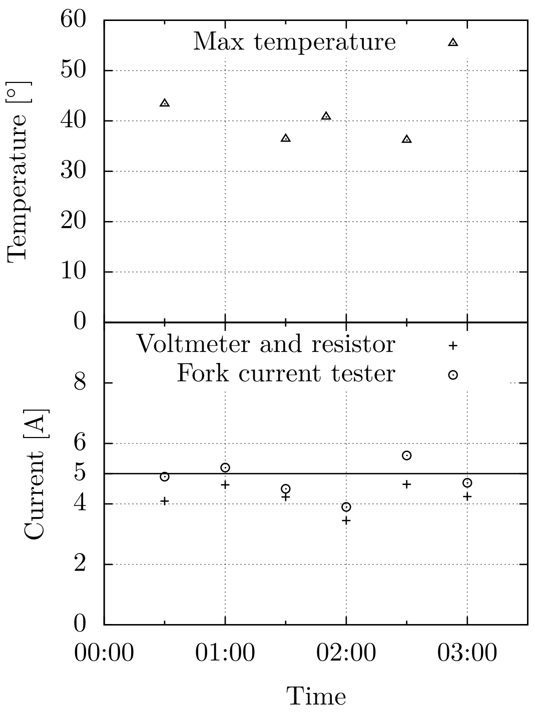


Start range: 100, stop range: 2000
Start ping: 0, stop ping: 800

Figure 1.36: Recording from one of the test surveys conducted on lake Římov. Noise is visible as horizontal lines.



(a) Current and maximum temperature during the 20 A temperature test. We attempted to keep the current at a constant 20 A. The max temperature was caused by the screw terminal, and does not give any information about the performance of the H-bridge.



(b) Current during the 5 A temperature test. We attempted to keep the current at a constant 5 A. The max temperature was caused by the series resistor for one of the indicator LEDs.

Figure 1.37: Current measured during the temperature test.

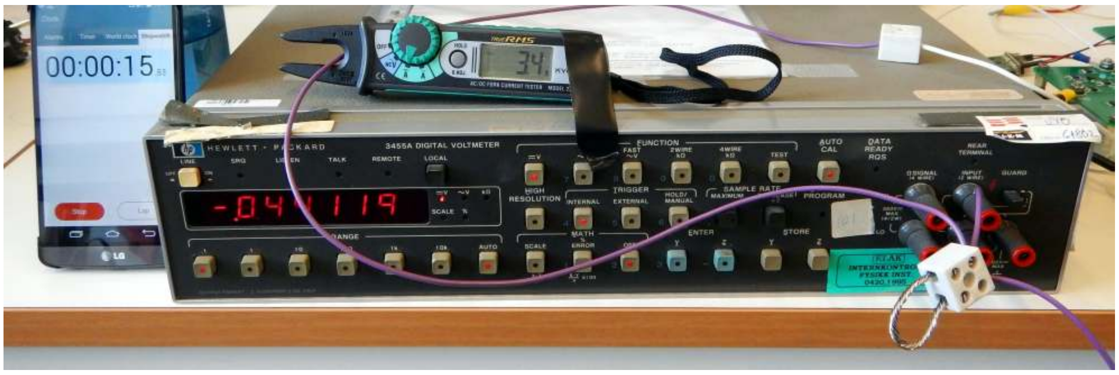
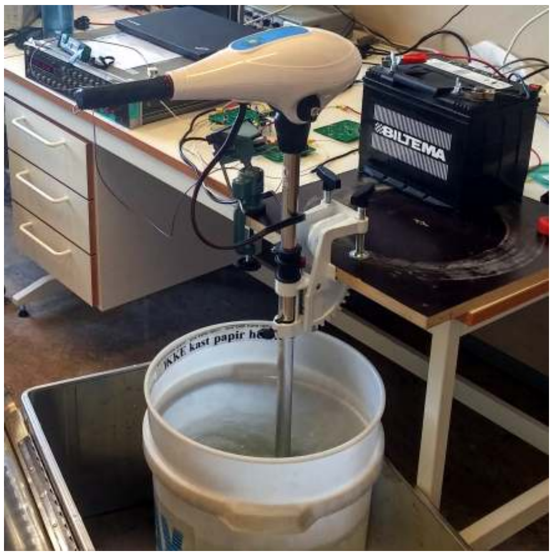
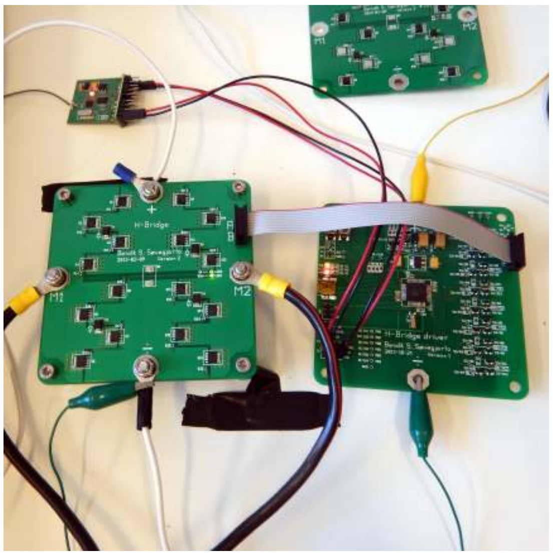


Figure 1.38: Current measurement setup for the temperature test. The power resistor is visible to the right.



(a) Electric outboard engine in a bucket of water, used as the load.



(b) H-bridge and H-bridge driver.

Figure 1.39: Temperature test setup.

1.5 Discussion

The following paragraphs discusses design choices, test results, problems encountered and possible enhancements for the development and use of the final motor control system. Potential future work is presented in the last sections.

1.5.1 Motor control system reliability

The motor control system has been successfully used in Lysaker and on lake Římov. Not a single second version H-bridge has died on its own over the course of the two years it has existed. Although multiple H-bridge PCB have become defective as a result of a user error. E.g. short circuit by a foreign object or shoot-through during flashing (reprogramming) of the H-bridge controller MCU.

1.5.2 Motor switching noise

Testing conducted on lake Římov with the motor control system and an echo sounder showed that the switching of the motor current created a lot of noise (visible in figure 1.36). To minimize this noise multiple measures can be taken; *a*) the motor cables should be kept away from the transducer cable; *b*) separate batteries should be used for the echosounder and the motor control system; *c*) the switching frequency can be changed to not be a harmonic of the echosounder frequency; *d*) the motor cables can be shielded; *e*) ferrite chokes can be added to the motor cables; or *f*) the whole motor control system can be placed further away from the echosounder. The motor cables were lying across the transducer cable during the recording on lake Římov.

1.5.3 Discussion of temperature test results

The resistance of the power resistor used to measure the current may have changed, as the high current running through it may have altered its temperature. We therefore trust the fork current tester instead, for the high current (20 A) test.

A bad connection between the cable and the cable crimp at the top screw terminal led to

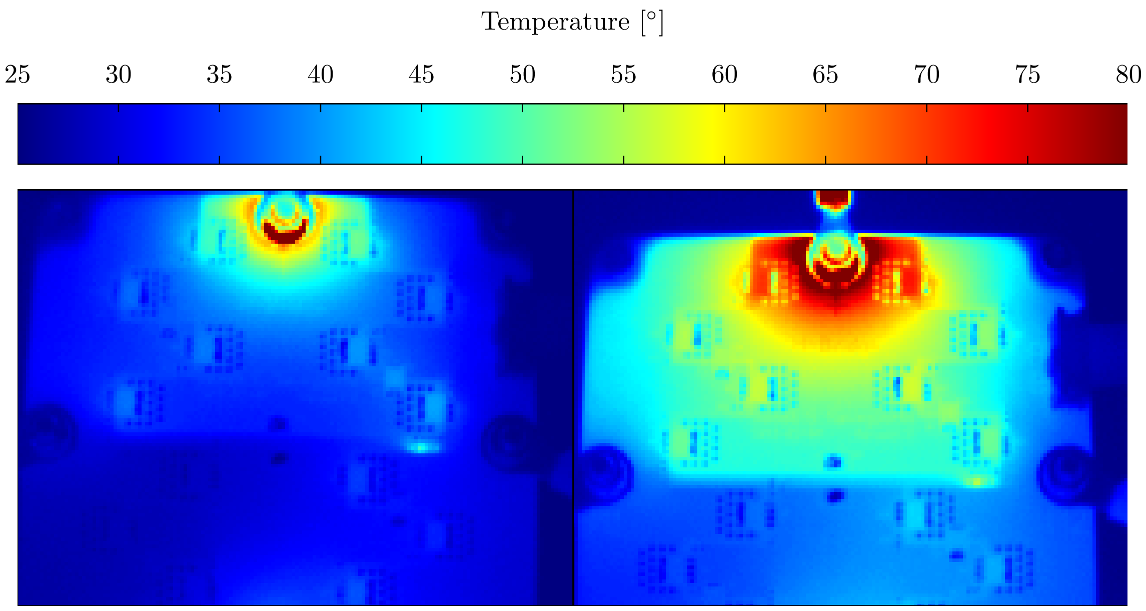


Figure 1.40: IR thermography of the 20 A temperature test. The left image shows the temperature after one minute, while the right image shows the temperature after three and a half minutes. A loose connection resulted in high thermal activity at one of the screw terminals.

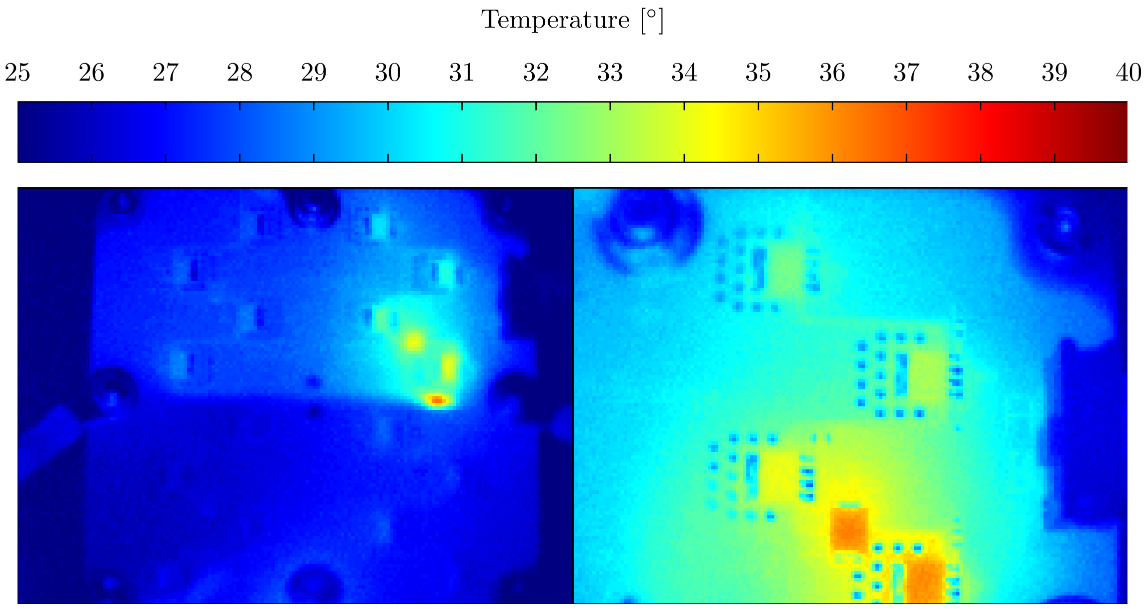


Figure 1.41: IR thermography of the 5 A temperature test. The left image shows the temperature of the whole H-bridge (after one minute), while the right image shows a close-up of the upper right corner (after two minutes). A series resistor for one of the indicator LEDs and a flyback-diode shows thermal activity.

high resistance and high thermal activity for the 20 A temperature test. This obscured the temperature monitoring of the rest of the H-bridge, but showed that parts of the PCB has good heat distribution and that the thermal vias are working. The heat distribution can be seen in figure 1.40, where the temperature difference is small over a large area, even outside the top copper plane. Figure 1.27 shows the location of the copper planes.

Thermal images from the 5 A test shown in figure 1.41 reveals multiple interesting points. The series resistor for one of the indicator LEDs is the hottest component on the PCB. This is caused by the high voltage drop over the resistor (560 Ω), and the relative high current consumption (18 mA) of the LED (governed by the resistor). The result being a higher power dissipation (180 mW) than the small 0603 resistor (100 mW) can handle. The resistance should be increased to decrease the current through the LED and resistor. Alternately, a physically bigger resistor must be used, e.g. 1206 (250 mW).

The upper left and lower right leg (corner) is used to conduct the drive current to the motor (in the forward direction). We expected the transistors in these legs to be the hottest components in the H-bridge. They are, however, only slightly hotter than the surrounding PCB.

The upper right leg gets hotter than the other legs of the H-bridge, and the flyback diode in this corner gets hotter than the transistors. The flyback diode in this leg conducts the back-EMF from the motor back to the power supply (capacitor and battery), when the drive current is removed. The constant voltage drop over the diode combined with this current results in a significant power dissipation. We believe the heat originating from the diode (and resistor) increases the ambient temperature around the transistors in the upper right corner, making them appear hot as well. Switching the current direction (through the H-bridge) should cause the upper left corner to become the hottest, and this could be investigated in a new test. Decreasing the dead-time may help offload the diode even more, and reduce its heat dissipation.

The motor cables were switched, so we drove the motor backward (when we thought we drove it forward). Driving it forward will result in a higher current consumption.

1.5.4 Stress test

The temperature test was originally planned to be a stress test, running the H-bridge until something broke or it reached a steady-state with constant temperature. This was unfortunately not possible, as the battery ran out, and there was not enough time for a new attempt. I also planned to perform multiple tests, with different H-bridges. Some with a different number of transistors, and some with a different type. For instance just using four transistors, or using the higher drain-source voltage version PSMN7R0-60YS, which have a higher $R_{ds_{on}}$. A new stress test, using an ample power source, could be performed by others if the maximum ratings of the H-bridge is needed.

1.5.5 Power dissipation

We have not yet measured and calculated the actual power dissipation of the assembled (final) motor control system. This was not deemed important, as the system functioned well during field use and managed to drive the engines (and boat) we required. If needed, the static power loss can be found by measuring the $R_{ds_{on}}$ or the voltage drop over the transistors and using equation (1.1) or (1.3) (with a constant load/current and switching disabled). Similar, the power loss during the switching transitions can be estimated by measuring the switching-transition time (Δt) and using equation (1.11).

1.5.6 Cable connections and -thickness

One of the nuts used to fasten the motor cable (through a bolt) to the H-bridge loosened during one of the tests on lake Římov. This resulted in the bolt getting very hot. This was later discovered as the bolt had permanently changed its color. Lock washers have since been used to prevent nuts from loosening. A bad connection was also encountered while conducting the temperature test. It is important to securely attach any wires used for conducting the large currents.

Appropriate wire thickness should also be used in a production setting. Some of the wires used when testing the H-bridge (and shown in some of the pictures) were relative thin. Connecting multiple wires in parallel was considered, but no noticeable temperature rise was found when using only one.

1.5.7 Power source

The Biltema engines states an operating time of 80 – 120 minutes at full speed and up to 4 hours at half speed, when using a standard 75 Ah lead-acid battery (for each engine). Multiple high capacity batteries may be connected in parallel to increase the operating time. A battery bank may not be feasible if very long surveys needs to be conducted. A diesel generator can then be used to supply power to electric engines.

1.5.8 Abandonment of HIP4081

The problems we experienced using the HIP4081 could perhaps be fixed with a RC-snubber suppressing the transients occurring at the load. By developing our own H-bridge controller and -driver, however, we were left with total control over the whole system and we had no problems with transients whatsoever.

1.5.9 Jump start problems

Earlier, we experienced a short jump start of both motors when the RC-transmitter was turned on after the motor control system had been connected to power. In other words, both engines tried to run at full speed for a fraction of a second the moment power was connected to the motor control system. We were therefore cautious to always turn the RC-transmitter on before power was applied to the motor control system. We suspected that the cause of this behaviour was that the MCU had its pins configured as inputs or low outputs during power-up. Pull-up resistors as described in the section 1.5.11 were thought to be a possible solution. In the end, we discovered that it was the RC-receiver which outputted invalid commands (pulse-widths) during power-up. By skipping the first RC commands (as described in section 1.3.2), the jump start problems disappeared. The motor control system can now be turned on, without problems, regardless of whether the RC-transmitter is turned on or off.

1.5.10 TVS diode

We did test the use of a TVS diode (SMCJ12CA) on the second H-bridge, but it did not seem to make any difference. This was however not examined in detail, and we did not use TVS-diodes further, since we did not have any problems with transients anymore (when using our H-bridge driver).

1.5.11 Drive-stage pull-up resistors

Pull-up resistors should be added on the control lines from the MCU to the drive-stage. Since these resistors are currently missing, the MCU must have its IO pins configured as outputs at all times. If they are configured as inputs, the control line will be left floating and could be registered as a low signal by the drive-stage. This would then turn «on» one of the transistors in the H-bridge, and possibly create a shoot-through condition. Pins are fortunately always configured as outputs (as long as the firmware is not modified), but pull-up resistors should be added nonetheless as a safety feature.

1.5.12 Number of power transistors

We initially decided to use four power transistors in each leg of the second H-bridge to distribute the heat and the electric current. One power transistor alone is, however, rated high enough to withstand the current drawn by our electric engines. It could therefore be possible to reduce the cost of the system, by reducing the number of power transistors. A stress test (as described earlier) may dictate if this is feasible.

1.5.13 PCB heat and current distribution enhancements

We only used a 2-layer standard PCB with a copper thickness of 35 μm for the H-bridge PCBs. If higher heat and current distribution is needed, the copper thickness can be increased and/or more layers can be used.

1.5.14 Firmware reorganization

The firmware for the second steering- and H-bridge controller (listed in appendix G) is currently doing all its calculation in interrupt service routines. The time used handling a interrupt should be kept minimal, as not to block other interrupts from being handled. The priority of the interrupts are programmable, so the interrupt with the long interrupt handler should at least have a low priority. An improvement could be to set a flag when a RC or UART command is received, and let the main program do the calculations currently performed in the timer interrupt. The timer could instead be used to wake the MCU from sleep, if lower power consumption is wanted.

1.5.15 UART communication

The UART steering command protocol does not include any error-detection system, since loosing some commands are not critical. When connecting or disconnecting another control system (e.g. an autopilot) over UART I have experienced that some steering commands (bytes) have been misread. A simple 1-bit Cyclic Redundancy Check (CRC) parity bit or a new protocol similar to NMEA 0183 (with checksum) can be implemented to combat these problems.

1.6 Summary

A functional motor control system has been developed, consisting of a RC-receiver (communication), a steering controller, two H-bridge controllers with drive-stages and two H-bridges. It is designed to drive two arbitrarily selected 12 V brushed DC electric motors at up to 40 A, although the max current rating has not been tested. The motor control system can also be used as a foundation for an autopilot. The use of a digital steering controlled proved to work better and give greater capabilities than an all analog version. We experienced problems when using a pre-made H-bridge driver, but a self-developed H-bridge controller and -driver worked successfully. A temperature test indicates that the power MOSFETs do not get very hot during normal use. Testing of the control unit (assembled motor control system), on two different boats with two different sets of engines, has shown that the motor control system is capable of controlling the electric engines we need. It has also shown that it is possible to use the equipment available, and that the complete cost of the system is low. The steering mechanism of combining both sticks provided very good control of the vessel, with a small boat (8 feet) giving especially high maneuverability. Noise problems caused by the motor switching were encountered during fieldwork, and these must be addressed before the motor control system can be truly usable for hydroacoustic work. Nevertheless, testing of the control unit together with a Simrad EK60 echosounder and a set of Ubiquity PicoStations show that the vessel can be used as a remotely controlled work platform.

Paper 2

Construction of an Attitude and Heading Reference System (AHRS)

Abstract

This paper presents the construction of an Attitude and Heading Reference System (AHRS) useful for hydroacoustic work. The AHRS outputs calculated tilt, roll and heading measurements based on sensor fusion from a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer. These measurements can be used to stabilize hydroacoustic recordings, or as a part of the autopilot for an Automatic Survey Vessel (ASV). Three attempts were done to create a functional AHRS, and both hard- and software for each of these will be described. The final attempt resulted in a unit which fulfilled our requirements of accuracy and reliability.

Keywords: AHRS, IMU

2.1 Introduction

This paper presents the construction of an Attitude and Heading Reference System (AHRS). The AHRS was needed for the following three tasks.

As part of the autopilot in my thesis covering the development of a «full-size low price Automatic Survey Vessel (ASV)» I need a compass to steer the vessel. A compass measures the magnetic field of the Earth, and outputs its heading relative to the magnetic poles. We wanted to create an AHRS which could function as a tilt-compensated[1] compass.

Horizontal hydroacoustic surveying is very susceptible to angular movement, since a small displacement at the face of the transducer results in a large shift far from the transducer. This movement can be caused by surface waves or human activity in the vessel conducting the survey. To compensate for this displacement, the angular movement can be recorded and a stabilization algorithm can be applied in post-process. This works similar to the optical image stabilization currently available in various consumer cameras. We wanted to create an AHRS to record the angular movement, and do so with a (tilt) accuracy of at least $\pm 0.5\,°$. This corresponds to a displacement of about $\pm 25\,\text{cm}$ for a target (e.g a fish) located 30 meter away from the transducer.

[1]A tilt-compensated compass does not need to be oriented parallel to the magnetic field for it to calculate the correct heading.

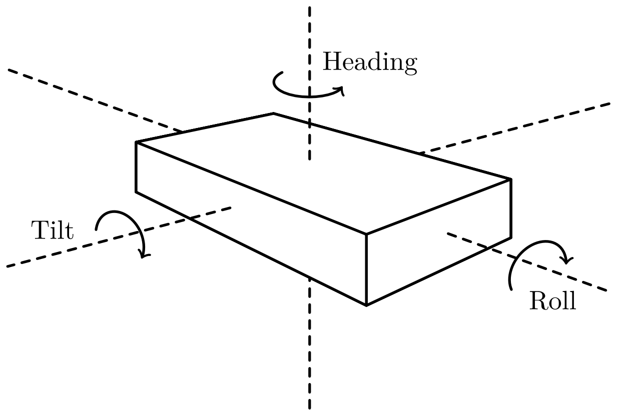


Figure 2.1: Illustration of tilt, roll and heading angles.

For some hydroacoustic experiments it is crucial to know the true tilt of the transducer. For instance when investigating surface-interaction during horizontal beaming. We wanted to create an AHRS which could function as an inclinometer, providing a reliable and accurate static orientation of hydroacoustic equipment.

We will describe how we solved these challenges by combining data from a 3-axis magnetometer, a 3-axis accelerometer and a 3-axis gyroscope.

Tilt, roll and heading angles are shown in figure 2.1.

A fellow master student, Johan Kleiberg Jensen, was at the time I needed a compass, working on his master thesis «Attitude Estimation for Motion Stabilization in Sonar Systems» [15]. He used a sensor with an embedded magnetometer. Unfortunately, he did not manage to read the magnetometer-data from the sensor, so I continued his work. Not only to get a compass for my own autopilot, but also to create an improved unit for experiments planned at lake Římov (České Budějovice, Czech Republic) the summer of 2013, where information about accurate tilt was essential.

The sensor Jensen found, contained three microelectromechanical systems (MEMS), a 3-axis accelerometer, a 3-axis gyroscope and a 3-axis magnetometer, all in one package. It is produced by InvenSense and called MPU-9150 (cost; ~8 USD each at a quantity of 100). The sensor is actually a combination of two chips, a MPU-6050 accelerometer and gyroscope and an AK8975 magnetometer. The gyroscope does not measure angular displacement, but angular velocity. The term gyroscope is thus technically not correct, and the term angular rate sensor would be more fitting. Since the term gyroscope is widely used when referring to an angular rate sensor, we will still use this term throughout the paper. Some specifications for MPU-9150 is listed in table 2.1. Current consumption varies depending on features enabled, sample rates, and sleep modes. More details about the MEMS can be found in [15–17].

Three attempts on creating a functional AHRS have been made. These are described in section 2.3. I have not developed any new algorithms for the sensor fusion, but merely adopted the work of others to create a device that fulfilled our requirements. Our main criteria was to get a unit which is *a*) low cost; *b*) sufficiently accurate; *c*) watertight; *d*) has good repeatability; and *e*) can calculate the heading. Jensen's unit did not fulfill criteria *e*) and *d*) (with our clone of his unit).

x-IMU from x-io Technologies[2] is an inexpensive AHRS platform, which could have been used instead. We chose to make our own because we wanted to continue the work of

[2]`http://www.x-io.co.uk/products/x-imu/`

Table 2.1: Key specifications for the MPU-9150.

Parameter	Specification
Gyroscope full-scale range	±250 °/s, ±500 °/s, ±1000 °/s, ±2000 °/s
Gyroscope sensitivity	$131\,\frac{\text{LSB}}{°/\text{s}}$ at ±250 °/s
Gyroscope total RMS noise	0.06 °/s
Accelerometer full-scale range	±2 g, ±4 g, ±8 g, ±16 g
Accelerometer sensitivity	$16384\,\frac{\text{LSB}}{\text{g}}$ at ±2 g
Accelerometer total RMS noise	4 mg
Operating temperature range	−40 °C – 85 °C
Operating voltage range	2.375 V – 3.465 V

Jensen and create a custom solution using a single integrated circuit (IC).

The AHRSs developed in this paper are named HydroAHRS, although they were initially known as «IMU» (Inertial Measurement Unit) or anglemeter.

2.2 Theory

2.2.1 Angle representation

There are multiple ways of representing spatial orientation, and Euler angles, quaternions and rotation matrices are widely used for this purpose. They are just briefly mentioned in the following paragraphs. An in-depth presentation can be found in [18].

Euler angles describes a spatial orientation using three parameters, each representing an elemental rotation about one of the axes in a 3-dimensional Euclidean space. Euler angles suffers from singularities (gimbal lock) around its poles.

Quaternions is an extension of complex numbers, making non-singular representation of spatial orientation and rotation possible. It is in other words capable of representing all rotations and angles, even around the poles.

Rotation matrices is another way to represent spatial orientation relative to a reference-axis set. They can avoid the singularities around the poles, but is more computational expensive than quaternions as they contain more parameters.

2.3 Material and methods

In order to find the orientation in space, data from all three sensors must be combined. The gyroscope is used to measure the angular velocity and find the angular displacement, while the accelerometer and magnetometer provides a reference. Angles calculated by simply summing the angular displacement (gyroscope), would suffer from a large drift over time. This method can also just calculate relative angles. To suppress the drift and achieve absolute angle measurements (referenced to the Earth), the accelerometer and magnetometer is needed.

Jensen developed a complimentary filter to combine the accelerometer and gyroscope data. This sensor fusion calculated the tilt and roll angels. I wanted to calculate the heading as well.

The first attempt on my own AHRS was to port Jensen's complimentary filter to an 8-bit microcontroller unit (MCU), instead of running it on top of a full operating system (OS). This port is described in section 2.3.1. The port was successful, but failed to fulfill our requirements of good repeatability. This was experienced during fieldwork at lake Římov, described in section 2.4.1.

I then gave it a second attempt using firmware given by the manufacturer of the sensor (InvenSense). This firmware enables the sensor itself to do the accelerometer and gyroscope fusion in its embedded Digital Motion Processor (DMP). Fusion of magnetometer data was based on work done by a company called Pansenti[3]. This implementation is described in section 2.3.2. The result was good tilt and roll measurements, however the heading calculation was unreliable (drifting and incorrect).

While struggling with the magnetometer fusion, InvenSense released a motion processing library which promised to do fusion of all its sensors. Accelerometer and gyroscope fusion was still done in the DMP, while the magnetometer data fusion was done in a pre-compiled library running on a MCU. Since I needed a fully functional compass for the Automatic Survey Vessel (ASV), I restarted the project yet again. Now based on both new hard- and software. The new software is described in section 2.3.3. The new hardware incorporated a 32-bit ARM Cortex-M MCU, which could handle the sensor fusion at the maximal rate of the sensor, 200 Hz.

The HydroAHRS outputs its calculated angles in the same (NMEA 0183 similar) format as originally used by Jensen [15]. The format is shown in figure 2.2 and table 2.2. The data format theoretically allows 71 bytes (maximum NMEA 0183 sentence length) to be output up to 200 Hz using 115200 baud over UART. A binary format, however, should be used if high throughput is needed, to reduce the amount of data transferred. For instance the binary protocol created by InvenSense to communicate with its OpenGL cube visualizer.

As the development of the firmware progressed, several hardware revisions were needed. The enclosures were also changed, to compromise between ease of assembly and water resistance. I will first present the development of the firmware, and afterwards the development of the hardware. Calibration and client side software will be described at the end.

2.3.1 First software attempt – Complimentary filter

The AHRS was originally based on the work of Jensen [15]. Jensen used a Raspberry Pi single board computer (SBC) running GNU/Linux to talk to the sensor and run his complimentary filter. I felt the use of a SBC with a complex OS was overkill for such a relatively easy task. I wanted to create an alternative implementation that was *a*) physically smaller; *b*) easier to make watertight; *c*) more power efficient; *d*) even more low cost; and *e*) easier to build multiple units of. I therefore decided to port his code so it could run on a 8-bit MCU. I chose an ATxmega AU-series MCU from Atmel, because it had all the peripherals I needed and was easy to work with. Atmel's toolchain is based on the GNU Compiler Collection (GCC) toolchain, which is very mature and well supported on multiple platform, including GNU/Linux[4].

His sensor fusion was built around a complimentary filter and showed promising results in testing [15]. I cleaned and restructured his single code file into appropriate functions over multiple files. I also wrote support code for Inter-Integrated Circuit (I^2C) and UART communication. I used a Bus Pirate[5] to debug the I^2C communication. Jensen's code is listed in [15, p. 99]. My code is shown in appendix J, licensed under GNU General Public License (GPL) version 2 or later.

I assembled my own prototype version to develop on, as described in section 2.3.4. I also built my own setup with an Raspberry Pi for reference, due to problems occurring while porting the code.

The AHRS is controlled by single character commands over a serial link (115200 baud), as listed in table 2.3.

One challenge I faced was the use of floating point numbers. I consider changing all calculations to use integer numbers, or take advantage of the implementation of fixed point math in version 4.8 of `avr-gcc`. Eventually, testing showed that the MCU could handle the floating point calculation at the low sample rate of 20 Hz.

I use a real-time clock (RTC) crystal of 32.768 kHz to calibrate the internal radio control (RC) oscillator and to timestamp samples with low drift.

[3]`https://www.linkedin.com/company/pansenti-llc`
[4]clarification; to develop on – I run Debian GNU/Linux on my workstation
[5]Bus Pirate: `http://dangerousprototypes.com/docs/Bus_Pirate`

```
      1          2      3 4      5      6      7   8   9  10 11 12
      |          |      | |      |      |      |   |   |   | |  |
$PASHR,hhmmss.sss,hhh.hh,M,rrr.rr,ppp.pp,xxx.xx,a.a,b.b,c.c,d,e*hh<CR><LF>
```

Figure 2.2: HydroAHRS data format.

Table 2.2: HydroAHRS data format field description.

Field	Placeholder	Description
1	hhmmss.sss	Time (relative)
2	hhh.hh	Heading [degrees]
3	M	Flag indicating true or magnetic north heading
4	rrr.rr	Roll angle [degrees]
5	ttt.tt	Tilt angle [degrees]
6	xxx.xx	Heave (not used)
7	a.a	Roll angle accuracy (not used)
8	b.b	Tilt angle accuracy (not used)
9	c.c	Heading angle accuracy (not used)
10	d	Aiding Status (not used)
11	e	Accuracy indicator
12	hh	Checksum

Table 2.3: Control commands for HydroAHRS mk.I.

Command (ASCII)	Command (hex)	Function
s	0x73	Start sending angles
S	0x53	Stop sending angles
g	0x67	Set gyro bias
a	0x61	Set angle reference
1	0x31	Send one measurement

2.3.2 Second software attempt – DMP

Since the complimentary filter did not give the level of response and repeatability that we needed (described in section 2.4.1) and because it was difficult to get a readout from the magnetometer embedded in the MPU-9150, I began looking for other sensor fusion solutions. InvenSense had released firmware which could be downloaded into MPU-9150 to enable its embedded Digital Motion Processor (DMP). The firmware is available for download at their website if you register as a developer[6]. A company called Pansenti had also developed a software library for Arduino and a library for Linux based platforms to work with the MPU-9150. The libraries included magnetometer data fusion. I chose to port the Linux based library to the ATxmega MCU I already used. The library was available at their GitHub[7] website in 2013, but has since been removed. The library was released under a MIT-license.

The code gets quaternions from the DMP and converts these into Euler angles before fusing them together with the magnetometer data. The magnetometer data is tilt-compensated and mixed together with the gyroscope heading to set the new heading angle.

After some struggling, I also managed to implement native Universal Serial Bus (USB) communication using the LUFA[8] library from Dean Camera (which had experimental support for ATxmega). This enabled me to use a Device Firmware Upgrade (DFU) bootloader to program (flash) the MCU over USB. The DFU bootloader is described in an application note from Atmel [19]. The HydroAHRS could now be uploaded with new firmware even after it had been casted in an enclosure.

The AHRS is controlled by single character commands, as listed in table 2.4.

The firmware for HydroAHRS mk.II is listed in appendix K.

[6]http://www.invensense.com/developers/

[7]https://github.com/Pansenti/linux-mpu9150

[8]http://www.fourwalledcubicle.com/LUFA.php

2.3.3 Third software attempt – MPL and DMP

Due to the problems described in section 2.4.5 regarding the non-functional and drifting compass, and other problems occurring with the use of Euler angles, I was looking for yet a new solution. I then discovered that InvenSense had released a library during the summer of 2014, which promised 9-axis fusion. This library is called Motion Processing Library (MPL) and is available for download if you register as a developer. It is pre-compiled and the source code is not available. They had some example code for use with an ARM Cortex M4 MCU and recommended a 32-bit CPU if 200 Hz sensor integration where needed. This would however mean I had to redo the project from scratch a third time. I chose an ARM Cortex M4 MCU from Texas Instruments (TI) named TM4C129 (cost; ~15 USD each at a quantity of 100), and decided to try again with ARM. This MCU was selected because it has hardware Ethernet PHY and MAC-layer embedded. The USB communication previously used has a short maximum length of only 5 m [20]. Ethernet enables up to 100 meter range [21].

Communication through Ethernet with TM4C129 has not yet been implemented for HydroAHRS, but has been done for the TM4C129 used in the autopilot described in paper 3. Although an Ethernet bootloader has been adapted and used successfully to program the HydroAHRS. For this, a separate BOOTP- and TFTP-server running on a nearby GNU/Linux computer was applied.

I had hardware kits from Texas Instruments (TI) which incorporated a MCU from the same family as TM4C129, the TM4C123. The TM4C123 is a smaller and cheaper version of the TM4C129 (without Ethernet). The kit was named EK-TM4C123GXL. I also had a matching sensor hub from TI, called BOOSTXL-SENSHUB, which had various sensors embedded, including the MPU-9150. TI also had code to do 6-axis fusion using their own

Table 2.4: Control commands for HydroAHRS mk.II.

Command (ASCII)	Command (hex)	Function
s	0x73	Start sending angles
S	0x53	Stop sending angles
r	0x72	Reset timestamp counter
1	0x31	Send one measurement
m	0x6D	Calibrate magnetometer
a	0x61	Calibrate accelerometer
M	0x4D	Reset magnetometer calibration
A	0x41	Reset accelerometer calibration
b	0x62	Run bootloader
w	0x77	Write mag. or accel. calibration
e	0x65	Read calibration data from EEPROM
i	0x69	Read serial number from EEPROM

complimentary filter. I decided to combine these two projects (from TI and InvenSense) into a functional 9-axis AHRS running on the TM4C123.

Receiving commands and saving calibration data to non-volatile memory is currently not implemented for HydroAHRS mk.III.

The firmware is unfortunately not listed in the appendix because of licensing restrictions, but is available in-house at the Department of Physics, University of Oslo.

2.3.4 Hardware for HydroAHRS prototype

The port of Jensen's work was initially developed on a prototype hardware platform. The prototype platform consisted of a veroboard with a breakout board for an ATxmega128A1 and a breakout board for the MPU-9150, shown in figure 2.3. Both were purchased from SparkFun[9].

2.3.5 Hardware for HydroAHRS mk.I

When the port of Jensen's code was functional I decided to design a printed circuit board (PCB) incorporating both the MCU and the sensor, along with necessary communication and support circuitry. The PCB consists of an 8-bit ATxmega32A4U, the MPU9150 sensor, status LEDs and additional components such as a crystal and various headers. The communication was done with UART from the MCU. Since I wanted to have USB connectivity for easy user handling, I used a FT232RL IC from FTDI as an UART-to-USB bridge. By doing so I was able to communicate and power the device using a single cable. It was also directly compatible with all new computers (which have USB). The FT232RL was also used as the power regulator. A standalone steel stencil was used to distribute solder paste for the Quad Flat No-leads (QFN)-package of the sensor. The PCB is shown in figure 2.4b and assembled in figure 2.4a.

The PCB was designed to fit into a watertight case. I chose a casted aluminium case from Deltron[10] because it was the smallest case I could find that would fit the PCB and a

[9]ATxmega128A1 breakout: `https://www.sparkfun.com/products/9546`, MPU-9150 breakout: `https://www.sparkfun.com/products/11486`

[10]Box: Deltron 483-0020

[11]IP68 certifies that the enclosure is dust tight and watertight in immersion beyond one meter of water.

[12]Connector: Bulgin PX0843/B, cable: Bulgin PX0840/B/5M00

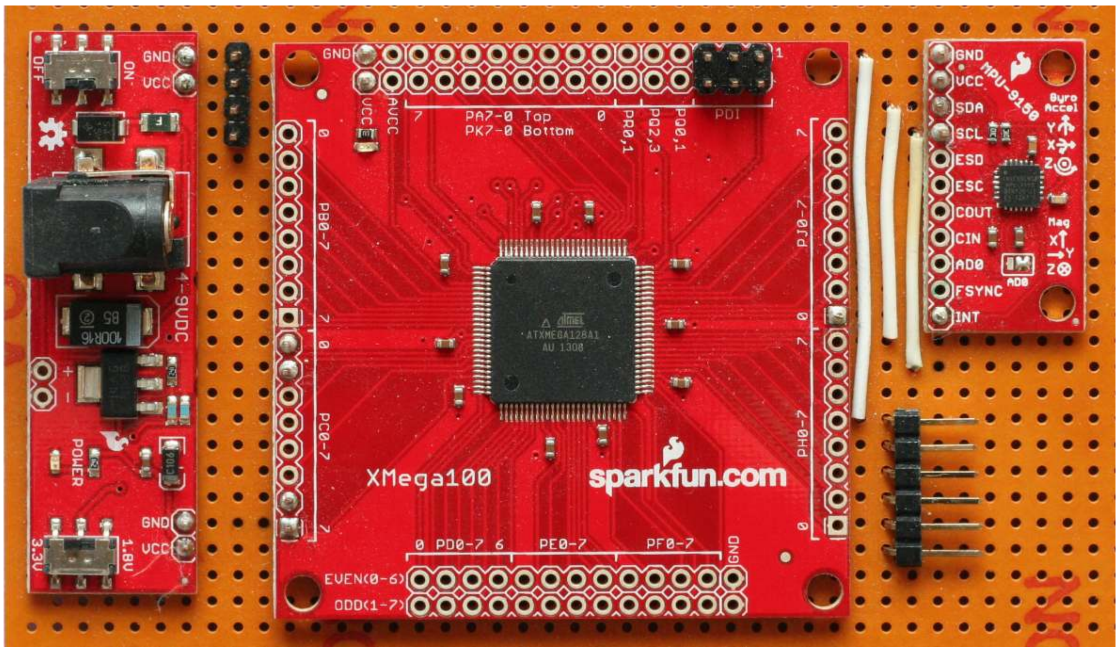
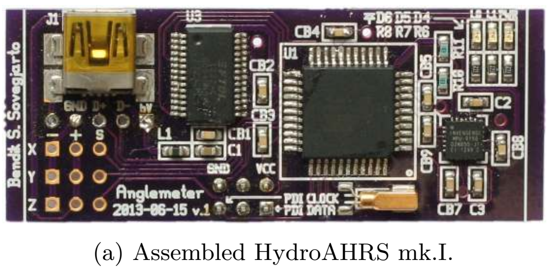
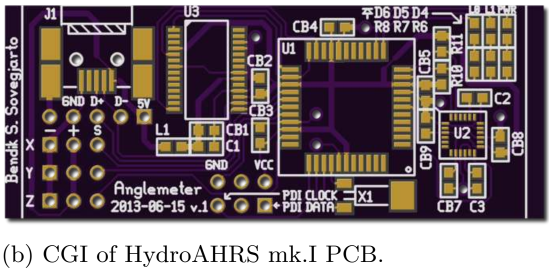


Figure 2.3: Prototype setup for porting the complimentary filter. In the middle is the ATxmega128A1 breakout board and to the right is the MPU-9150 breakout board.



(a) Assembled HydroAHRS mk.I.



(b) CGI of HydroAHRS mk.I PCB.

Figure 2.4: Images of HydroAHRS mk.I PCB.

USB connector while being rated IP68.[11] I found watertight USB connectors and cables from Bulgin[12].

I made two HydroAHRS mk.I units and the instrument workshop at the department of Physics helped me drill holes for the connectors. These units were built to examine a hydroacoustic phenomenon when doing horizontal beaming. One of the HydroAHRS mk.I units assembled in the enclosure is shown in figure 2.5a.

The gasket along the edges of the enclosure was covered in silicon grease to further increase the water resistance. The PCB was also placed away from the bottom of the enclosure, to allow some water intake to be tolerated. Nevertheless, fieldwork in Czech Republic showed that the casing and the connector was not watertight after rough handling. When the USB cable got a tug, water escaped through the gasket. Water was also leaking in along the edges of one of the enclosures, probably caused by the front plate being bent during drilling of the connector hole.

2.3.6 Hardware for HydroAHRS mk.II

For the HydroAHRS mk.II a new PCB was designed and two new enclosures were built.

HydroAHRS mk.II – PCB

The new PCB was designed to fit into the first of the new enclosures, and was quite compact. It had the Future Technology Devices International (FTDI) chip removed as I now used the MCU for native USB communication. I added an external power regulator and electrostatic discharge (ESD)-protection for the USB signals. The PCB is shown in figure 2.6b and assembled in figure 2.6a.

A total of eleven HydroAHRS mk.II has been assembled.

HydroAHRS mk.II – First casing revision

Due to the problems with the box used by HydroAHRS mk.I, I wanted a smaller enclosure with shorter edges. I also wanted to fill the enclosure so there was no place for the water to go. Since the USB connector was creating difficulties, we opted for a non-removable cable instead. I built and used my home-assembled CNC-milling machine and milled acrylic plastic sheets in different shapes, as shown in figure 2.7a. Multiple shapes were tested, and a version of the final shape is shown in figure 2.7b. I wrote a Python script to create the G-code needed to mill this shape.

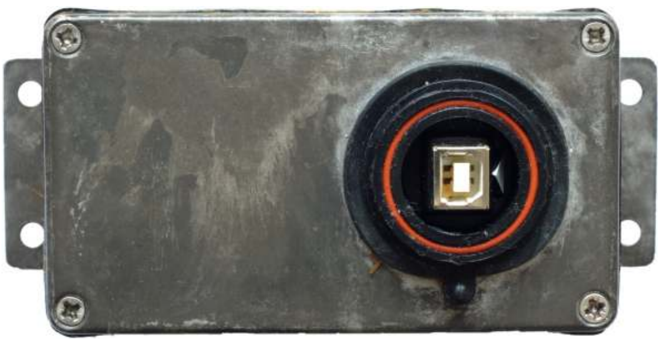
The PCB with components was placed in one of the milled sheets. The USB cable was wrapped with self-amalgamating tape to fill the stress relief chamber. The main chamber was filled with liquid epoxy. The other sheet was placed on top, with a coat of acetone on all connecting faces. The acetone dissolved the surface of the acrylic sheets and fused the two sheets together. An assembled unit with this casing is shown in figure 2.5b. This enclosure proved to be watertight, but required a complicated manufacturing process and as later testing have indicated, was prone to inflicting ESD damage.

HydroAHRS mk.II – Second casing revision

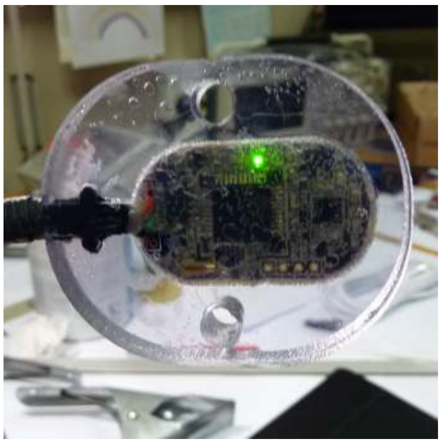
A new revision of the enclosure for the same PCB was developed using another casted aluminium case. We went back to using aluminium as this would safely conduct static discharge from other objects away from the PCB. The casing would not create a static field itself neither. Since the USB connector was removed, the case could be quite compact. The PCB's mounting brackets and the USB cable was submerged in Tec 7[13], to attach the PCB to the case and provide stress relief for the cable. Instead of using epoxy to fill the air cavity, we used canning wax. Later on, the wax can be melted away if the PCB needs reparation. Self-amalgamating tape was used for cable bend protection.

The process of mounting HydroAHRS mk.II in the aluminium enclosure is shown in figure 2.9

[13]MS polymer based sealant and adhesive, `http://www.tec7.ie/products/tec7-sealant`

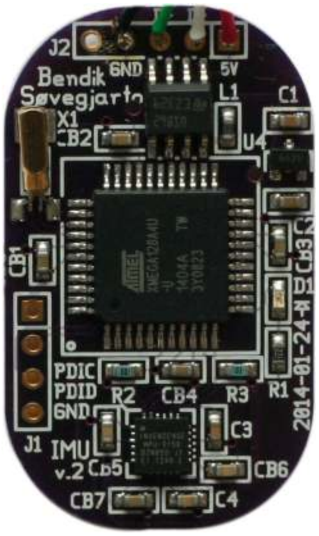


(a) Enclosure for HydroAHRS mk.I.

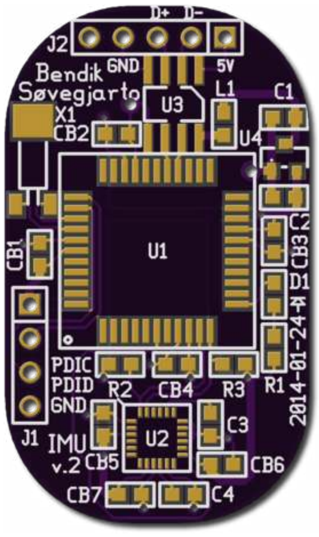


(b) HydroAHRS mk.II in the first revision casing.

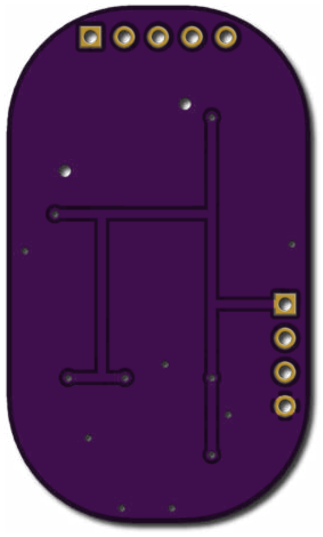
Figure 2.5: Enclosures for HydroAHRS mk.I and mk.II.



(a) Assembled HydroAHRS mk.II.

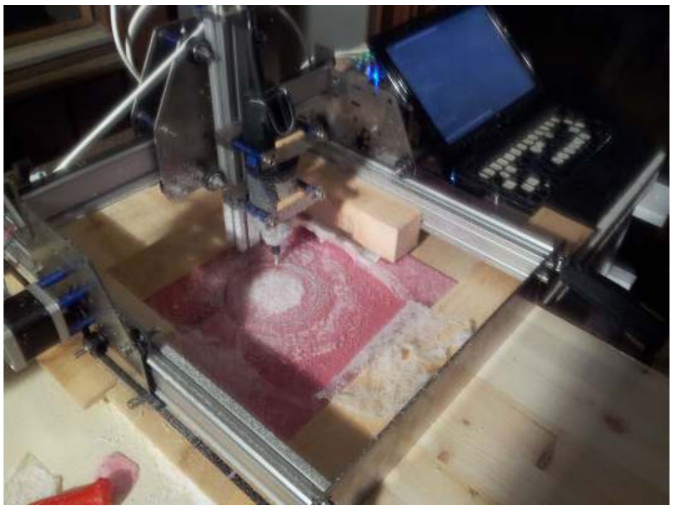


(b) CGI of HydroAHRS mk.II PCB top side.



(c) CGI of HydroAHRS mk.II PCB bottom side.

Figure 2.6: Images of HydroAHRS mk.II PCB.


(a) eShapeOko milling acrylic sheets.

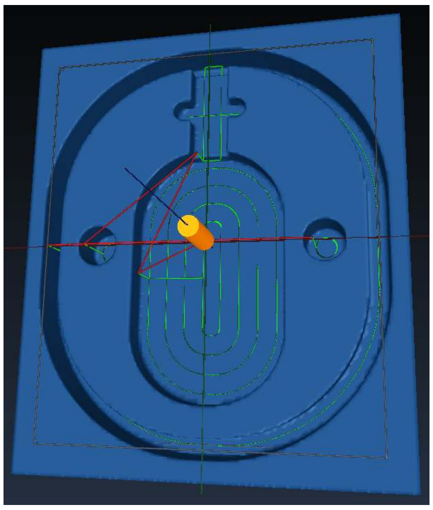

(b) Final shape of HydroAHRS mk.II (first revision) casing.

Figure 2.7: Milled HydroAHRS mk.II (first revision) casing.

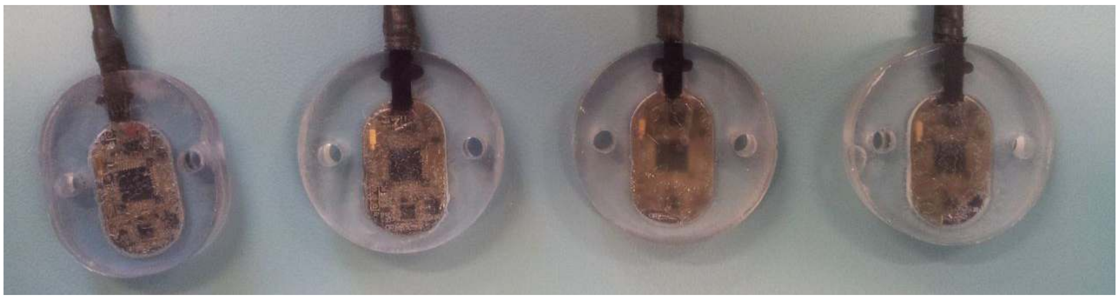
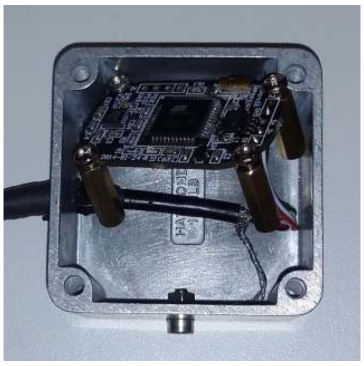
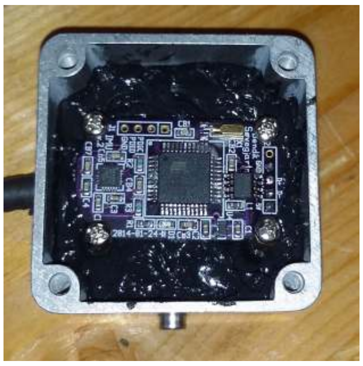


Figure 2.8: Four HydroAHRS mk.II assembled in their (first revision) casings.


(a) HydroAHRS mk.II in aluminium enclosures, before mounting.


(b) HydroAHRS mk.II in aluminium enclosures, after being submerged in Tec 7.

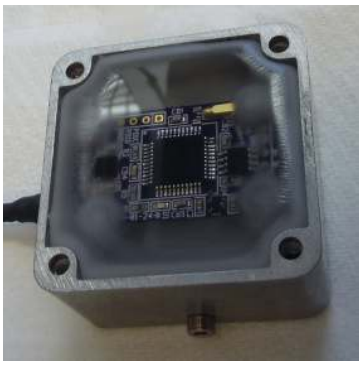

(c) HydroAHRS mk.II in aluminium enclosures, after being filled with wax.

Figure 2.9: The mounting process for HydroAHRS mk.II, with the second revision casing.

2.3.7 Hardware for HydroAHRS mk.III

With the change to a new MCU architecture, yet another hardware revision was needed. The new PCB incorporates the TM4C1294NCPDT from TI, the MPU9150 sensor, a new power regulator, SD-card connector, Ethernet connector, status LEDs and support circuitry. The SD-card connector is added for future use, if logging to SD-card is needed. Since the new MCU has embedded Ethernet MAC and PHY layer support, Ethernet has also been added as a possible communication option, with passive Power-over-Ethernet (POE) as the power supplier. The other option is UART (possible to convert to RS-232 or a current loop) with 5–24 V power.

The PCB is shown in figure 2.10b and assembled in figure 2.10a.

A total of 13 HydroAHRS mk.III has been assembled.

The PCB is designed to be mounted on a metal plate, with a protective layer moulded over it to make it watertight. The mould will have a metal mesh inside to protect the PCB from ESD damage. This enclosure has not yet been built or tested.

2.3.8 Calibration

Multiple calibration methods has to be applied for the HydroAHRS to be accurate. Each of the three sensors must be calibrated individually, and this has been done in different ways for the various HydroAHRS revisions. The tilt offset and slope error is then calibrated.

Magnetometer calibration

The HydroAHRS mk.I did not use the magnetometer, so no calibration was necessary.

The HydroAHRS mk.II magnetometer calibration is initiated by the user. The user then moves the unit around, while the MCU finds the maximum and minimum raw magnetometer measurements for all three axes. This is stored in EEPROM, and the subsequent raw magnetometer measurements is offset and scaled according to the mean and range (spread) of the max./min. values. The spread of the raw magnetometer measurements after calibration from a HydroAHRS mk.II is shown in figure 2.11.

The HydroAHRS mk.III uses the MPL to automatically calibrate the magnetometer when a good figure-eight movement is detected. This is currently not saved to a non-volatile memory, and is lost after a power-cycle.

Accelerometer calibration

The HydroAHRS mk.I did not calibrate the accelerometer.

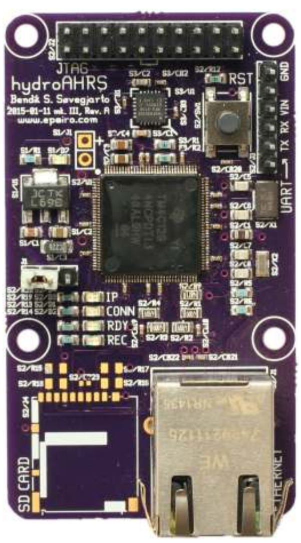
The HydroAHRS mk.II accelerometer calibration works similar to its magnetometer calibration. The unit is moved around, and the maximum and minimum accelerometer measurements are found. This must be done with very slow movements, as not to apply erroneous acceleration. We only want to measure the gravitational acceleration of the Earth. The max./min. values are stored to EEPROM, and the subsequent raw accelerometer measurements is offset and scaled accordingly.

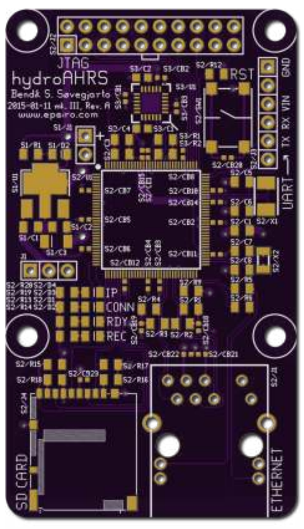
The HydroAHRS mk.III uses the MPL to perform a self-test and calibrate the accelerometer. The unit must be face-up or face-down during the calibration. This self-test is at the moment automatically performed on power-up or after a reset. The calibrated values is saved to (non-volatile) registers in the MPU-9150. The self-test can be modified to only run after a user request (and make use of the non-volatile nature of these registers), if a command interface for HydroAHRS mk.III is implemented.

Gyroscope calibration

The HydroAHRS mk.I uses a very primitive method to calibrate the gyroscope. A single measurement is taken while the unit is motionless, and this is subtracted from all subsequent measurements to remove the bias.

Both HydroAHRS mk.II and HydroAHRS mk.III uses the DMP to automatically calibrate the gyroscope if it senses that it has been motionless for eight seconds. HydroAHRS mk.III also runs a self-test on power-up or after a reset, which calibrates the gyroscope. It must therefore be motionless on power-up.


(a) Assembled HydroAHRS mk.III.


(b) CGI of HydroAHRS mk.III PCB top side.

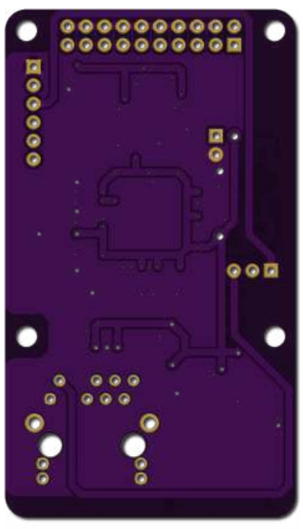

(c) CGI of HydroAHRS mk.III PCB bottom side.

Figure 2.10: Images of HydroAHRS mk.III PCB.

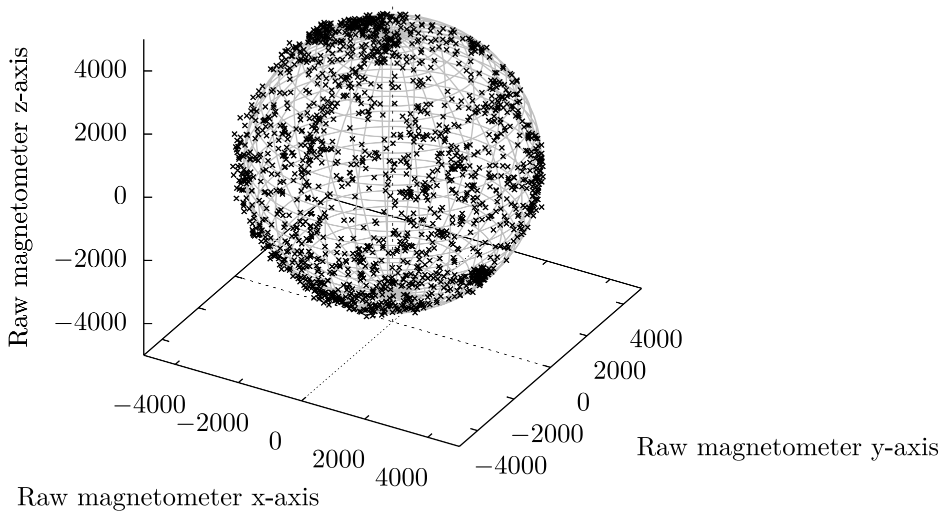


Figure 2.11: Calibrated raw magnetometer measurements from HydroAHRS mk.II.

Tilt offset calibration

There exists an offset error of every angle calculated from a HydroAHRS, originating from offsets in the IC assembly to the PCB, and the PCB mounting to the case. We are most interested in getting rid of the tilt offset. The total tilt offset can be found by placing the unit parallel to a true vertical (or horizontal) reference. The difference between the calculated tilt and the true tilt of 90 (or 0) degrees, is the tilt offset. The HydroAHRS mk.II was calibrated against a vertical reference as described in section 2.4.4.

Tilt slope calibration

To correct for the relative error between each tilt measurements, the slope must be calibrated. If the tilt is changed from 10 to 20 degrees, we expect the unit to change just as much as if it was changed from 40 to 50 degrees. The tilt measurements should be linear and have a slope of 1. To check if the tilt measurements are linear and find the slope, the unit is tilted while a different reference setup measures the relative tilt. The slope of several HydroAHRS mk.II units were found using the setup described in section 2.4.4.

2.3.9 Front-end software

Multiple different software solutions has been used to communicate with HydroAHRS.

Serial terminal emulator

GNU `screen` has been widely used as a serial terminal emulator during development. Under GNU/Linux you can simply write `screen /dev/tty<device> 115200` to connect to a HydroAHRS at 115200 baud (press ’s’ and start receiving data from HydroAHRS mk.I or HydroAHRS mk.II). The data can be logged to file with `cat /dev/tty<device> >> log.nmea`.

Windows application

Helge Balk wrote a windows application in Pascall using Lazarus for communication, calibration and logging with HydroAHRS. The application is shown in figure 2.12.

Python- and web-based front-end

I created a web-based front-end in JavaScript/-HTML/CSS to visualize the data from HydroAHRS. A Python server talks to the HydroAHRS over USB and redirects the data to the web-app. The application is shown in figure 2.13.

Another Python-application using matplotlib for the realtime-plotting has also been developed.

InvenSense cube

InvenSense has released a Python application which renders the spatial orientation of an AHRS as a cube using OpenGL. The application uses a binary format to transfer data from the AHRS and can thus handle very high sampling rates.

2.4 Testing and results

The HydroAHRS mk.I and mk.II have been tested in both controlled environments and in fieldwork, however, the last version (HydroAHRS mk.III) has not.

2.4.1 Fieldwork in Czech Republic 2013

HydroAHRS mk.I was created for its use in fieldwork in Czech Republic at lake Římov in August 2013. The experiments conducted revolved around phenomenas occurring during horizontal beaming with hydroacoustic echo sounders. The mounting angle of the equipment had to be known for the experiment to be successful.

Two HydroAHRS mk.I units were built and brought to the Czech Republic. To test their water resistance they were suspended in a bucket of water for multiple hours. No leakage was observed.

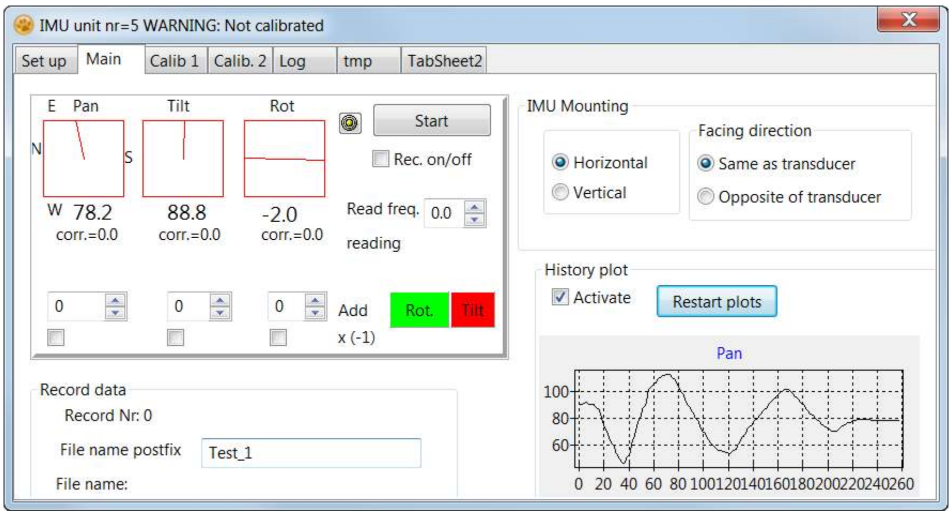
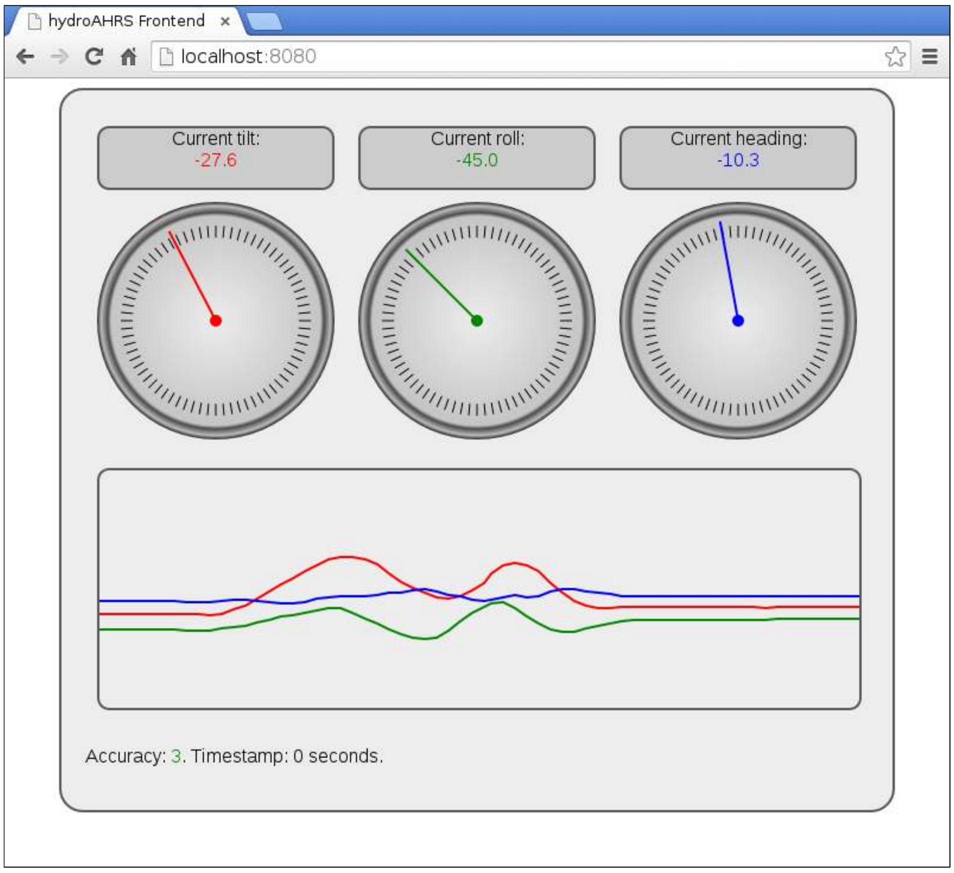


Figure 2.12: Windows-based front-end for HydroAHRS.



Current tilt:
-27.6

Current roll:
-45.0

Current heading:
-10.3

Accuracy: 3. Timestamp: 0 seconds.

Figure 2.13: Web-based front-end for HydroAHRS.

The units were then calibrated using a Leica DISTO D5 as reference. The Leica has an accuracy of $\pm 0.3\,^{\circ}$. It is shown in figure 2.14.

A bash script was used to record the tilt from the HydroAHRS mk.I and note the reference value from the Leica. Gnuplot was then used to plot the results and to fit the data to a linear function using a Non-linear least squares (NLLS) Marquardt-Levenberg algorithm (available in Gnuplot). The resulting linear regression gave the offset and slope error of the HydroAHRS mk.I compared to the Leica.

One of the units was mounted on the same plate as a Simrad EK60 $10 \times 4\,^{\circ}$ transducer, as shown in the bottom of figure 2.15a, after it had been calibrated. This unit was used to report the tilt angle during the installation and use of the transducer. 15 meter long active USB-extension cables were used to connect the HydroAHRS mk.I to a computer.

The other HydroAHRS mk.I was later mounted on a sub-Atlantic pan (heading) tilt rotator, along with a transducer, as shown in figure 2.15b. The rotator had motors inside, making it possible to turn it around in two axes and adjusting its heading and tilt. The current heading and tilt was reported by the rotator using an integrated feedback potentiometer. A Rieker H5A1-90 one axis inclinometer was also mounted on the rotator. The rotator was set to different tilt angles, and the same Leica DISTO D5 was used to measure the angle. The results are shown in figure 2.16.

During the experiments using the HydroAHRS mk.I, two problems were encountered. A slight drift in the tilt and roll measurements was discovered. This is most likely caused by the system slowly converging against a value. More importantly, it was discovered that the HydroAHRS mk.I had low repeatability. Meaning that a power-cycle would change the measurements even though the unit was stationary. This made it practically impossible to calibrate the device, and we did not trust its data. After multiple days of attempting to get trustworthy readings, a fail-safe replacement was used instead (of any of the electronic measuring devices). The replacement was a simple protractor, consisting of a string with a mass attached, an aluminium bar with degree markings and a webcamera.

2.4.2 HydroAHRS mk.II repeatability test

The repeatability of HydroAHRS mk.II was tested by mounting the device to a protractor. Multiple readings were then taken at various angles, with a power-cycle in between. The measurements showed that the same angle was found within $0.05\,^{\circ}$ every time, after waiting for the device to settle. This was, however, only tested with one unit, so until a more thorough repeatability test has been conducted, we regard the tilt repeatability to be within $\pm 0.1\,^{\circ}$.

2.4.3 HydroAHRS mk.II RTC drift test

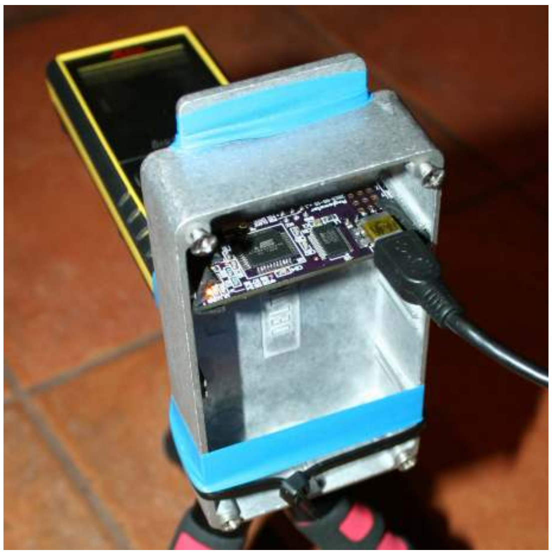
To find the level of drift on the timestamp calculated from the RTC crystal we let a unit run for multiple hours and compared the start and stop timestamp with a Network Time Protocol (NTP)-synchronized computer. We found approximately 13 seconds drift after 20 hours and 48 minutes. This corresponds to 174 ppm and is not adequate. A workaround is discussed in section 2.5.4.

2.4.4 HydroAHRS mk.II indoor test and calibration

Two procedures were conducted to find the offset and slope error of each unit, and to see how linear their response were.

HydroAHRS mk.II offset calibration

First the offset error was found by placing the unit parallel to a vertical surface. The vertical surface was found by aligning an U-shaped aluminium bar to a long string with a mass attached at the end. The mass was suspended in a bucket of water to dampen its movements. The long length of the string and bar made it possible to align the bar accurately. The setup is shown in figure 2.17a.


(a) Leica mounted to a unit under test (UUT) (HydroAHRS mk.I).

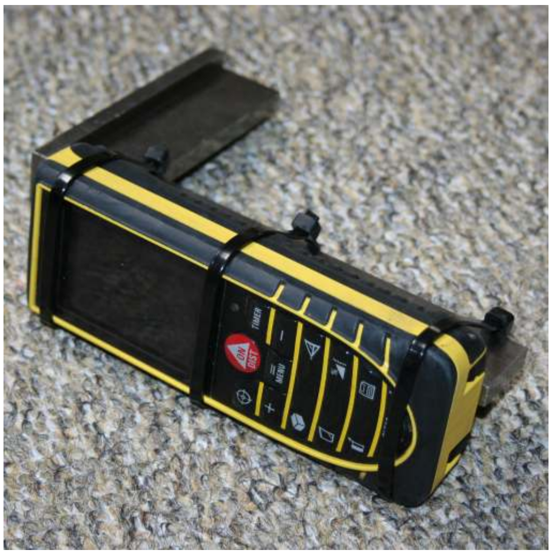

(b) Leica mounted to a try square for easy calibration.

Figure 2.14: Photographs of Leica DISTO D5 used for calibration.

(a) HydroAHRS mk.I during installation on transducer plate.

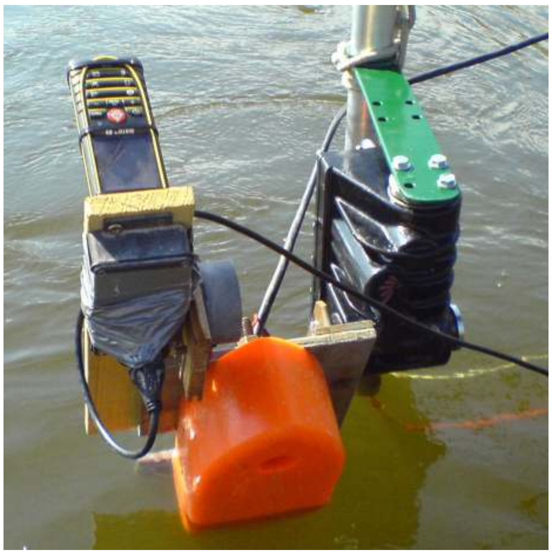

(b) Rotator (black), along with the transducer (orange), the HydroAHRS mk.I (square grey box), the Rieker (round grey box) and the Leica (yellow).

Figure 2.15: HydroAHRS mk.I in use at lake Římov in 2013.

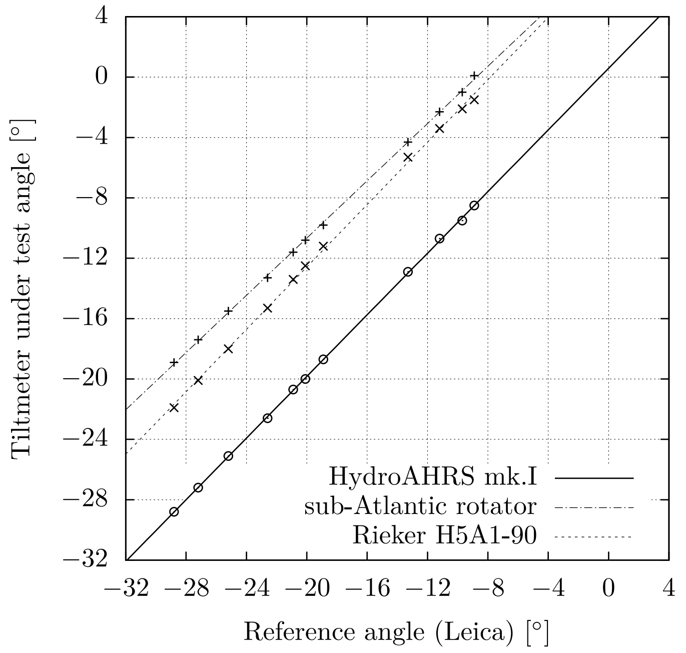


Figure 2.16: Calibration points and linear regression for tilt-meters used at lake Římov in 2013. The HydroAHRS mk.I, Rieker H5A1-90, and rotator-tiltmeter are shown.

HydroAHRS mk.II slope calibration

We used Pythagoras theorem to find the slope error. The unit under test (UUT) was mounted to an aluminium bar (figure 2.17b) at a known distance away from a vertical ruler (figure 2.17c). The ruler was confirmed to be vertical by using the same method as described in the offset-calibration. A laser pointer was attached to the end of the bar. By tilting the bar, the laser dot was moved between different steps of the ruler. The angle reported by the UUT was recorded for each step. This was done with the UUT mounted both normal and reversed (flipped front-to-back), to expose the offset error. The measured data is shown in figure 2.18.

2.4.5 Fieldwork in Czech Republic 2014

HydroAHRS mk.II (both with the first and second casing) have been tested by FishEcU[14] during fieldwork they have conducted in the Czech Republic.

Multiple units of the first revision (plastic) casing stopped working (no response over USB). We suspect they died because of ESD-damage. None of the second revision casing units have died.

FishEcU reported that the heading calculation drifted substantially. This has been verified by us. We found approximately 130 ° drift over 8 hours.

2.4.6 Dynamic test

A dynamic test setup has been created (together with Jensen), but the test has not yet been performed with any of the HydroAHRS. This test would show the response time of the system. An AHRS can be placed on a wooden rod with one end connected to a wheel, while the other

[14]Fish Ecology Unit of the Department of Fish and Zooplankton Ecology of the IH BC CAS. `http://www.fishecu.cz/`

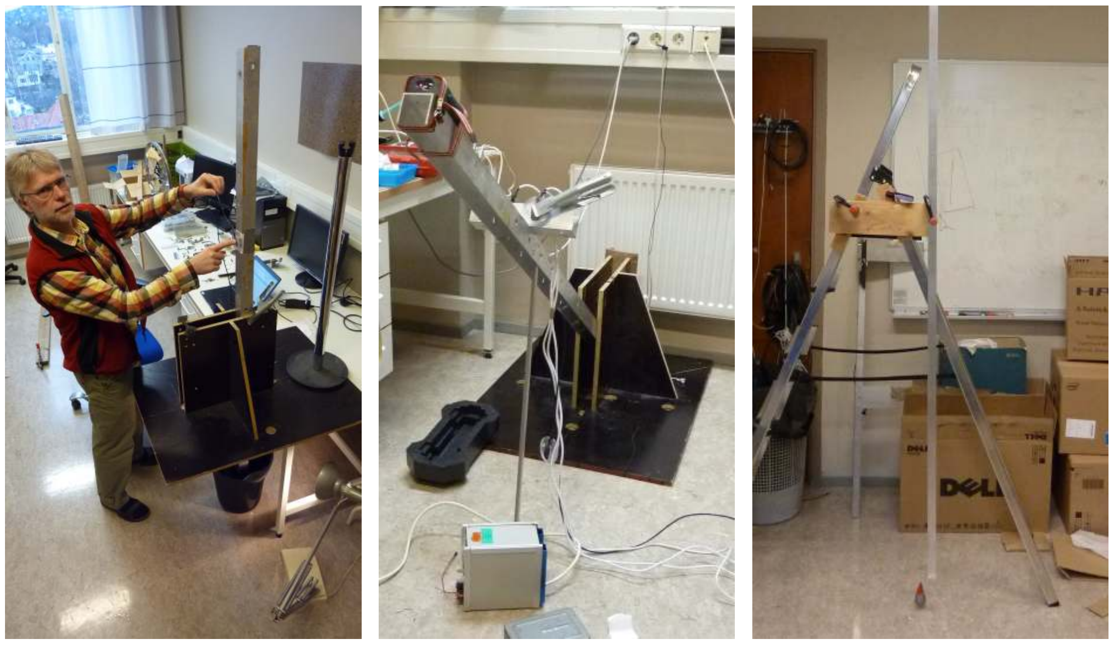

(a) Vertical reference, using a string and a heavy mass. (b) Aluminium bar with UUT. (c) Vertical ruler.

Figure 2.17: Offset and slope-calibration setup.

end of the rod gliding along the table. The wheel is shown in figure 2.20.

When the wheel is turned to an angle θ, it will move the rod to an angle φ, as shown in figure 2.21. The AHRS unit will measure the tilt angle φ directly and save it. Since we knew the geometry of the setup we can calculate φ from θ, by using equation 2.1.

$$\varphi = \arcsin\left(\frac{r\sin(\theta)+b}{L}\right) \tag{2.1}$$

A PCB I created, containing a Atmel AT-mega64 MCU (not ATxmega), is used to find θ. Between every other pair of stokes on the wheel, a piece of semi-reflecting tape is attached. This reflects the light emitted from an infrared (IR) LED, and was received by a photodetector[15]. The photodetector module output a series of pulses as the wheel is turned. The higher the revolution, the higher the frequency of these pulses. The MCU received the pulses and counts the time between them.

The maximum and minimum angle of the rod can be determined by using a laser pointer, pointing at the opposing wall. By measuring the distance between the two points as well as from the wall to the setup, simple trigonometry can be used to give the angle, as done in the slope calibration setup.

2.4.7 HydroAHRS mk.III indoor test and calibration

The HydroAHRS mk.III has not yet been tested or calibrated in a controlled environment. Although preliminary tests have not shown drift on the calculated heading. It has also shown that the unit is capable of handling high sample rates (up to 200 Hz). The tilt and roll calculations are based on the same firmware for the DMP as the HydroAHRS mk.II, and is thus expected to provide similar results.

[15]The photodetector modulated the IR LED, so it was quite resistant of disturbances from the surrounding light

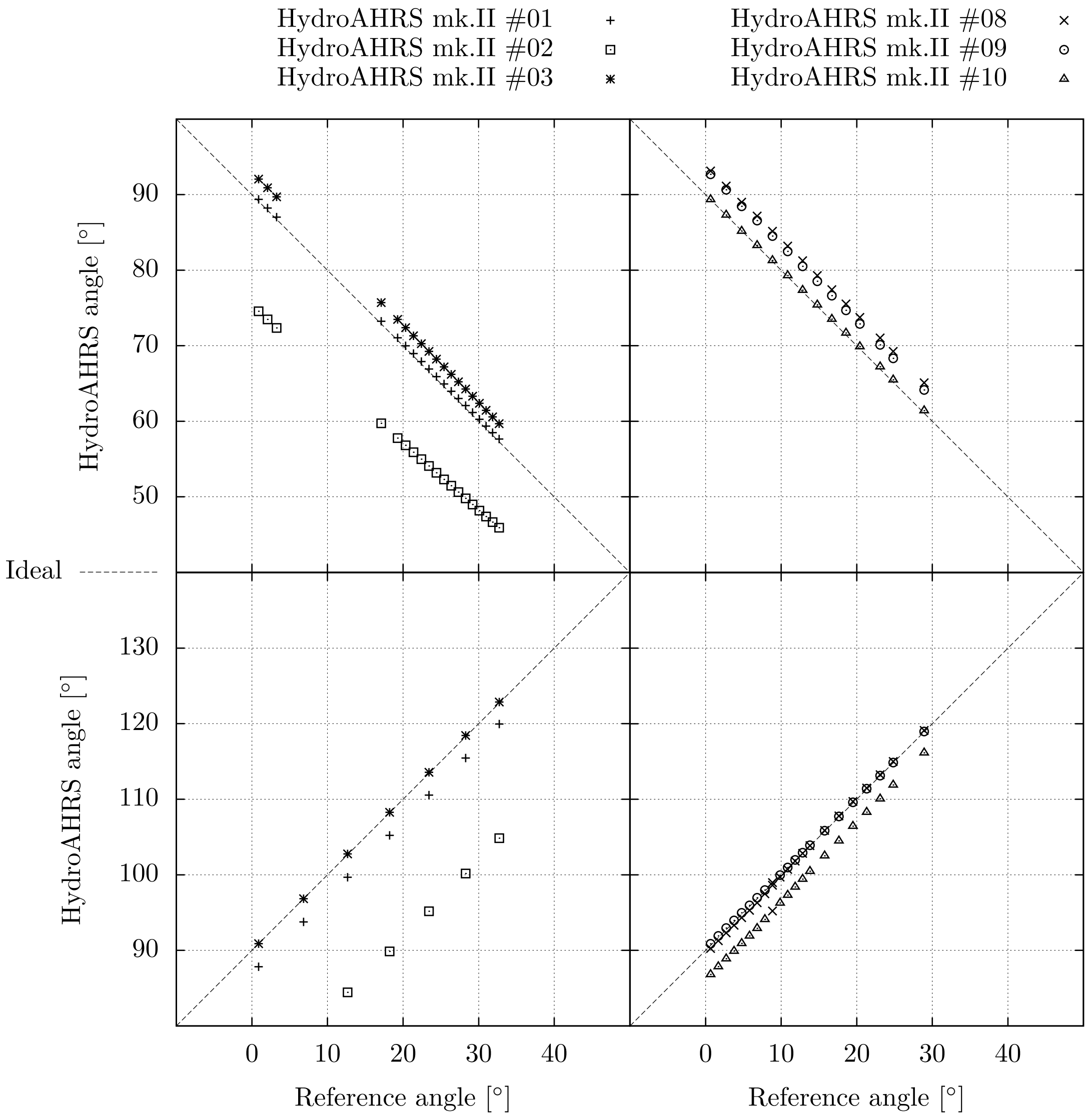


Figure 2.18: Slope calibration measurements taken with HydroAHRS mk.II units 1,2,3 in March 2014 and 8,9,10 in June 2014. Measurements were taken with the front side away from the ruler (top), and again with the front against (bottom). This shows the offset error, as can be seen with HydroAHRS mk.II unit 10. Its measurements follows the ideal line when facing away from the ruler, but misses the line after it has been flipped.

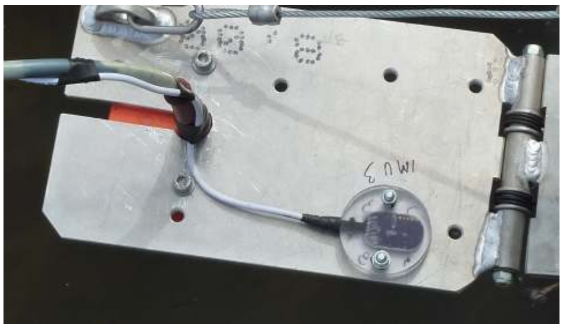


Figure 2.19: HydroAHRS mk.II (first casing revision), used in experiments conducted by FishEcU in the Czech Republic in 2014.

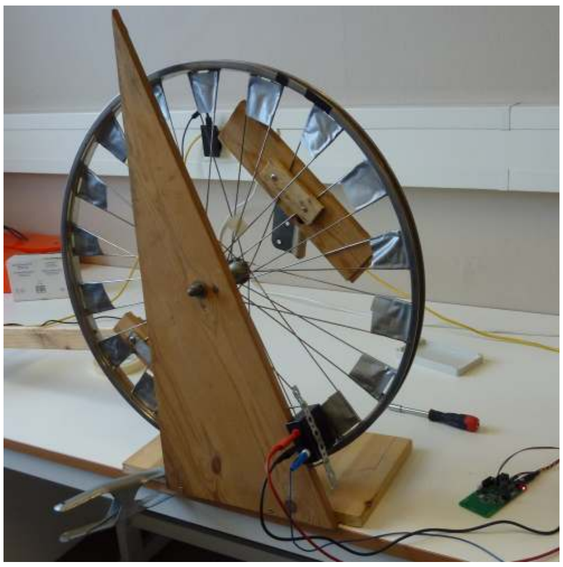


Figure 2.20: Photograph of the wheel and detector PCB for the dynamic test.

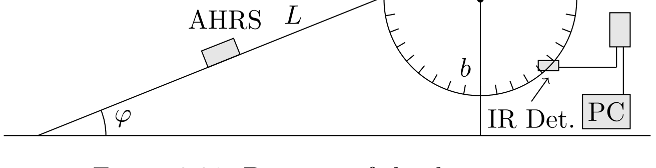


Figure 2.21: Diagram of the dynamic test.

2.5 Discussion

The following paragraphs discusses some thoughts regarding the development and use of the HydroAHRS.

2.5.1 HydroAHRS mk.II tilt test results

Figure 2.18 shows that the tilt-measurements from all the HydroAHRS mk.II units tested are linear over their working range. The repeatability test described in section 2.4.2 shows that power-cycles does not affect the measured tilt. The slope and offset error may thus be removed through calibration, which results in accurate tilt-measurements. The exact accuracy of the HydroAHRS mk.II (or mk.III) has not been determined, but an (absolute angle) accuracy near the repeatability of $\pm 0.1\,^\circ$ should be achievable through calibration.

2.5.2 Comparison test

It should be emphasized that testing of the complimentary filter, which showed bad repeatability was done on the HydroAHRS mk.I. A test comparing Jensen's unit and HydroAHRS mk.I was planned, but got cancelled when a new solution for the sensor fusion emerged.

2.5.3 Mounting orientation

Care must be taken when calibrating and mounting the units, to make sure the units are mounted the same way as when they were calibrated.

2.5.4 Timestamp drift

To circumvent the high drift on the timestamp from HydroAHRS mk.II, we can use the client computer to reset the RTC frequently. Another solution is to let the client computer ask for single samples and timestamp the sample on the computer. If the computer is also used for receiving the data from an echosounder, both will be timestamped with the same clock and automatically be synchronized. There is a delay (with jitter) between the clients request and a sample has been delivered. This delay and jitter should be measured.

2.5.5 DMP and MPL documentation

The firmware downloaded into the DMP of the MPU-9150 is a pre-compiled binary blob. With the lacking documentation of the DMP, it makes it very difficult to inspect what the DMP is actually doing or change its behaviour. Nevertheless, testing has shown that the DMP is performing a satisfactory sensor fusion of the accelerometer and gyroscope data. The same applies for the pre-compiled MPL, used for the magnetometer fusion.

2.5.6 Future work

Future work involve testing the heading calculation of the final version, testing the dynamic response at the full sampling rate of 200 Hz, implementing Ethernet communication and saving all calibration data to EEPROM.

2.6 Summary

Three different attempts have been made on designing a fully functional AHRS, with both software and hardware undergoing these revisions. These are named HydroAHRS mk.I, mk.II and mk.III. The final version (mk.III) uses a 32-bit ARM Cortex-M MCU, a MPU-9150 (3-axis accelerometer, 3-axis gyroscope and 3-axis magnetometer) sensor, and code supplied from the manufacturer (InvenSense) to perform sensor fusion and angle calculations. Fieldwork has shown that the HydroAHRS mk.II, with the second casing revision, can be successfully used under water. A simple repeatability test with power-cycles between each measurement was conducted with HydroAHRS mk.II. The test showed that the same angle was found within $\pm 0.1\,^\circ$ every time, after waiting for the device to settle. Offset and slope calibrations have been performed on six HydroAHRS mk.II units, which all showed a linear response over their working range. Together with the repeatability test, an accuracy close to $\pm 0.1\,^\circ$ should thus be achievable. We have managed to create a unit which fulfilled our requirements of *a*) low cost; *b*) sufficient accuracy; *c*) watertightness; *d*) repeatability; and *e*) with heading calculation, although the last version (mk.III) has not yet been tested.

Paper 3

Construction of a primitive autopilot for hydroacoustic work

Abstract

This paper presents the construction and implementation of a primitive autopilot used to automatically steer a work platform for hydroacoustic work. Together with a vessel, this creates an Automatic Survey Vessel (ASV). A Global Navigation Satellite System (GNSS) and a compass is used to steer the vessel at constant speed through multiple waypoints. Realtime transmission of new waypoints enables dynamic route generation. Recorded or pre-planned routes can also be executed. The autopilot has been successfully tested on a land-based rover.

Keywords: Autopilot, GNSS

3.1 Introduction

A lot of equipment and manpower is required to conduct hydroacoustic surveys with high coverage. Earlier work described in paper 1 presents the construction of a remote controlled hydroacoustic work platform. This work platform can be controlled from a distance, thus making it possible to do hydroacoustic experiments or fieldwork without letting movement from the operator disturb the hydroacoustic recordings.

To further increase the efficiency of hydroacoustic fieldwork, an autopilot can be used to control the vessel during surveys. This has multiple applications, for instance automatic night surveying, automatic abundance estimation, environmental surveillance, and as a tool for manual surveying. By telling the autopilot to follow a larger vessel with a crew, higher survey coverage can be achieved and complicated driving patterns can be created (as the small boat is more agile than the large manned vessel). This paper will describe the implementation of a primitive autopilot as a proof of concept for future work. A vessel with this autopilot is named an Automatic Survey Vessel (ASV).

The autopilot is able to follow a route consisting of multiple waypoints. These waypoints will be given to the autopilot while it is running. The waypoints can originate from a pre-planned route on a map, an earlier recorded route, or be dynamically calculated by a separate computer in realtime. This makes it possible to create driving patterns on-the-fly. For instance to make the ASV drive in patterns around a larger vessel with a crew. The autopilot will follow pre-programmed rules, without making any decisions while it is running. It is therefore not autonomous, but merely automatic. All activity will be under human supervision. A web-based control panel is used to observe and control the vessel, and communicates with the autopilot using WebSockets.

A similar project was presented in 2013 [22], focusing on automatic estimation of fish density. This is one of the possible application with our ASV. Other related projects include [3, 23–25].

3.2 Theory

3.2.1 PID-regulator

A PID-regulator is a simple yet powerful control loop system. It continuously calculates the error between the current state of a system and the wanted state (setpoint). The error signal is modified by the PID-regulator to produce an optimal control signal for the system, using multiplicative, integral and derivative terms, as shown in figure 3.1.

More information about PID-regulators is available at [26–28].

3.2.2 WebSockets

WebSockets is protocol that allows full-duplex communication between a web-browser and a server. The server can send messages to the web-browser without being polled for the information. WebSockets makes it possible to create real-time interactive control systems in a web-browser.

It has been standardized by the Internet Engineering Task Force (IETF) as RFC 6455. More details can be found in the standard [29].

3.2.3 High accuracy positioning

Multiple solutions are possible to get high accuracy position data. Primarily the setup of a personal Differential GPS (DGPS) system, subscription to correction data distributed by satellite or land-based networks or the use of real-time kinematics.

A local DGPS system can be created by placing one or multiple Global Navigation Satellite System (GNSS)-receivers around an area to be examined. These will be fixed to the ground and can average their position over time to accurately return their true position. A deviation from this position is an error, mainly created by atmospheric phenomena. This deviation (for each satellite) can be sent to a mobile vessel, enabling it to compensate for the error in its own measurements.

Multiple permanently installed base stations, often located along coastal shores for use in maritime navigation, make up already available DGPS systems. Access to the deviation data from these systems can be purchased, with a price depending on the level of accuracy needed (deci- or centimeter). The data is commonly relayed by radio from land stations. Kystverket [30] and Kartverket [31] provides such services in Norway. High accuracy is achieved when the distance to the base stations is short.

Deviation data can also be relayed using satellites (satellite-based augmentation system (SBAS)) and constitutes a wide area DGPS-system. The accuracy is not as a high as a localized DGPS system, since the base station is usually far away, but the coverage is good. European Geostationary Navigation Overlay Service (EGNOS) is a SBAS deployed in Europe and Wide Area Augmentation System (WAAS) is a SBAS covering North America.

Real-time kinematics measures the phase of the carrier wave from a GNSS to achieve high accuracy, in addition to using the information encoded within the GNSS-messages [32]. RTKLIB is a free[1] program package for GNSS positioning, making all the calculations necessary to perform real-time kinematics. RTKLIB needs access to raw data from a GNSS-receiver, which is only available on selected devices (e.g. U-blox 6 PPP).

Further information about GNSS and SBAS can be found in [33].

3.2.4 Haversine

The haversine formula is an equation which gives the great-circle distance between two points on a sphere, with good numerical precision [34].

With ϕ_1, ϕ_2 as the latitude of point 1 and 2, λ_1, λ_2 is the longitude of point 1 and 2,

[1]as in free speech

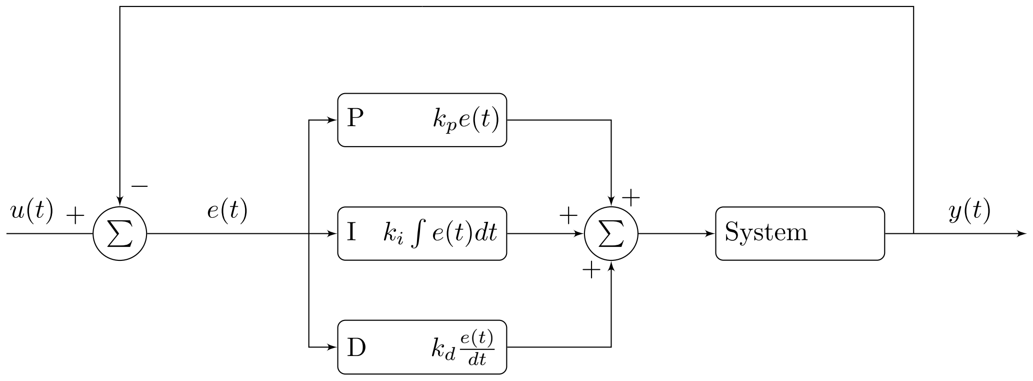


Figure 3.1: Overview over a PID-regulator.

$\Delta\phi = \phi_2 - \phi_1$ and $\Delta\lambda = \lambda_2 - \lambda_1$, the haversine is given as equation (3.1).

$$a = \sin^2 \frac{\Delta\phi}{2} + \cos\phi_1 \cos\phi_2 \sin^2 \frac{\Delta\lambda}{2}$$
$$d = 2r \arcsin\left(\sqrt{a}\right) \tag{3.1}$$

where r is the radius of the sphere, and d is the distance.

3.2.5 Bearing

The initial bearing between two points on a sphere is given by equation (3.2) [35].

$$y = \sin\Delta\lambda \cos\phi_2$$
$$x = \cos\phi_1 \sin\phi_2 - \sin\phi_1 \cos\phi_2 \cos\Delta\lambda$$
$$\theta = \arctan\frac{y}{x} \tag{3.2}$$

where θ is the bearing, ϕ_1, ϕ_2 is the latitude of the initial and final point, and $\Delta\lambda$ is the difference between the longitude of the initial and final point.

A normalized bearing (0–360 °) is given by $(\theta{+}360) \,\%\, 360$, where % is the (floating) modulo operator.

3.3 Material and methods

The autopilot extends the remote work platform described in paper 1. It will be placed as a fourth printed circuit board (PCB) on the control unit stack. A block schematic of the complete control unit stack is shown in figure 3.2, and a photograph in figure 3.3.

3.3.1 Selection of GNSS

We chose to use a single GNSS-receiver as a start, since this is the lowest entry point (simple and inexpensive) to a positioning system and is usable anywhere on Earth. The GNSS-receivers found have support for SBAS and this has been enabled most of the time. A single GNSS-receiver may be sufficient, as we do not have very strict requirements for absolute position. We will rely on the compass for short term navigation and by only using the autopilot in lakes (and away from the shore), we have few obstacles. Good coverage can also be achieved without having to follow an exact route. A more advanced positioning system can be implemented at a later stage, if needed.

We initially chose to use a Trimble Copernicus II, but preliminary testing showed disappointing accuracy and reception sensitivity (under difficult conditions). A U-blox 6 Precise Point Positioning (PPP) evaluation kit was acquired, since it promises sub-meter precision when used together with a SBAS. It can also output the raw data needed by RTKLIB.

Non-PPP variants within the same family have been used afterwards, namely U-blox NEO-

[2]CEP is a measure of precision. It sets the radius of a circle (with the mean as center), where 50 % of all samples is expected to be included.

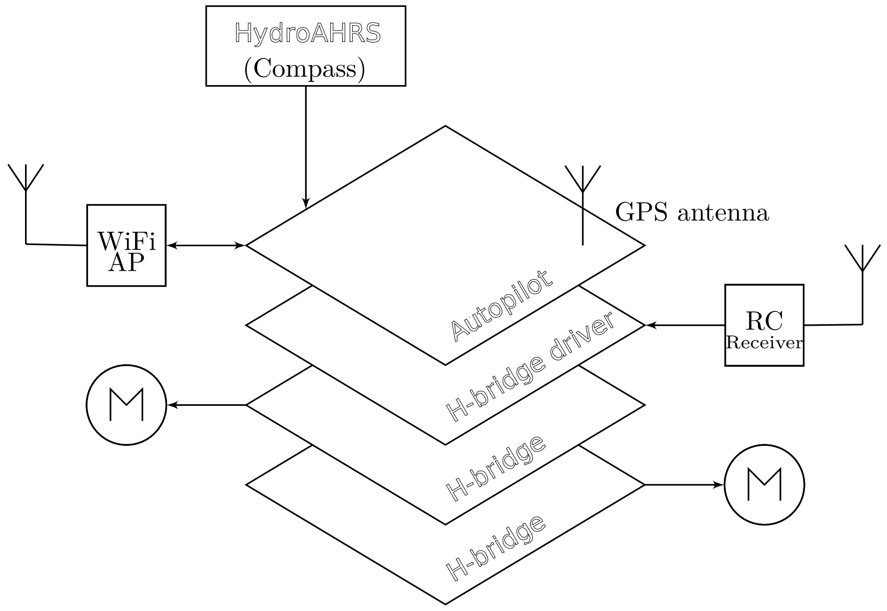


Figure 3.2: Block schematic of the complete control unit stack.

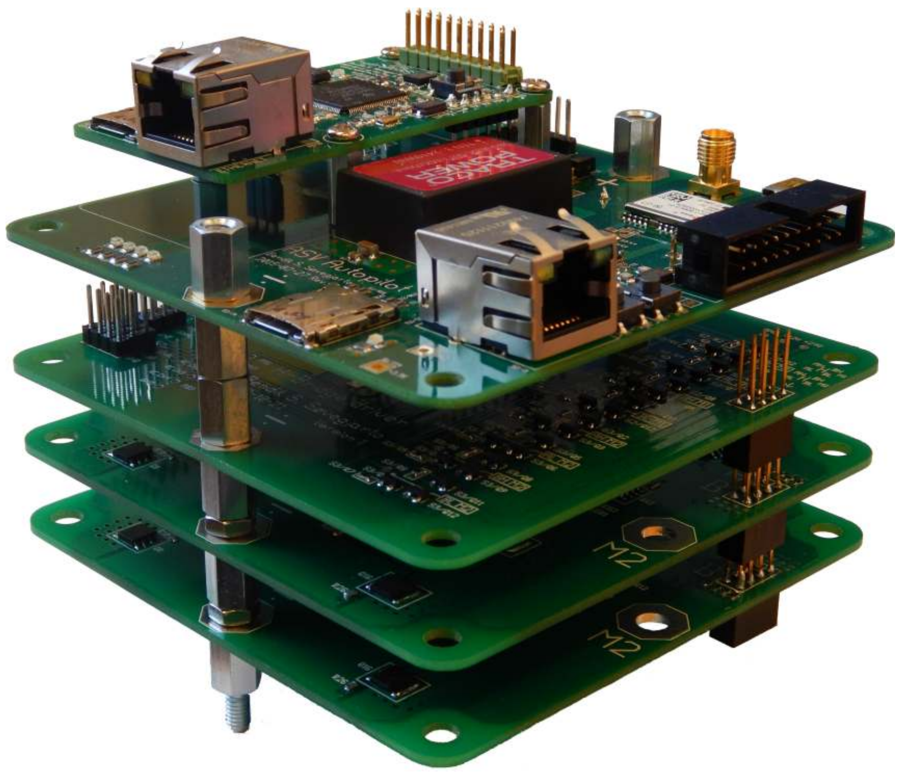


Figure 3.3: Photograph of the complete control unit stack, with motor control system, autopilot and compass.

6M, U-blox NEO-7M and U-blox NEO-M8N. U-blox NEO-M8N promises a position precision of 2.0 m Circular Error Probable (CEP)[2] and has been chosen as the GNSS-receiver to use in the autopilot. This is the latest device in the U-blox NEO M-series and offers a good compromise between price (44 EUR) and performance.

We have used active (GPS only) patch antennas supplied by U-blox during testing and development. As suggested by [36], changing to a geodetic-grade antenna is an effective way to increase performance.

Three tests investigating the performance of the chosen GNSS-receivers have been conducted, and are described in section 3.5.

3.3.2 Selection of compass

Along with a GNSS to get position data, the autopilot will use an Attitude and Heading Reference System (AHRS) to get the current heading. Our self-made HydroAHRS described in paper 2 will be used as the compass.

3.3.3 Selection of MCU

ATxmega32A4U from Atmel was initially chosen as the microcontroller unit (MCU) for the autopilot, but this was later changed to an ARM Cortex M4 MCU from Texas Instruments (TI) named TM4C129 (cost; ~15 USD each at a quantity of 100). The change was done because TM4C129 has both Ethernet MAC and PHY layer embedded, making it possible for it to talk directly over Ethernet.

OpenOCD was used together with a USB JTAG debugger[3] from Olimex to flash (program) the ARM MCU. GNU Debugger (GDB) was used together with OpenOCD to debug the MCU.

3.3.4 Firmware and autopilot algorithm

A very simple algorithm is used in this autopilot. Based on the measured location of the vessel (from the GNSS), and where we want arrive (from the waypoint), the wanted course is calculated using equation (3.2). The measured heading (from the compass) is then used to calculate the thrust needed to achieve the wanted heading corresponding to the calculated course. The heading is adjusted while maintaining (near) constant speed. A PID-regulator is used to calculate the thrust needed to achieve the wanted heading, as fast as possible with minimum overshoot. It is based on an application note and code from Atmel [28]. A waypoint is approved as passed when the GNSS-receiver reports a position within a perimeter around the waypoint's center. The next waypoint will then be used as the new destination, or the autopilot will stop if no new waypoints are available. New waypoints received by the autopilot will be acknowledged back to the sender upon arrival.

Firmware incorporating this algorithm was written for the TM4C129. A block schematic of the autopilot algorithm and program flow is shown in figure 3.4. The firmware for the autopilot is listed in appendix M.

3.3.5 Hardware and PCB

The hardware of the autopilot consist of the main MCU which calculates the thrust to each of the engines, a GNSS, a compass and a communication unit. All of these modules were separate PCBs during the development of the autopilot, as shown in figure 3.5.

These components were finally combined into one 4-layer PCB, with the compass externally connected as a snap-in board. To minimize the signal loss from the Global Positioning System (GPS) antenna to the GNSS-receiver special care was given during the routing of the RF-signal path. The trace width was adjusted together with the substrate thickness to give a matched impedance of 50 Ω. A separate solid ground plane is placed under the GNSS-receiver circuit, connected to the digital ground plane in a single point. A guard ring is routed around the GNSS signal path to minimize noise reception.

The PCB is shown in figure 3.6 and an assembled autopilot is shown in figure 3.7.

[3]Prod. id: ARM-USB-OCD-H

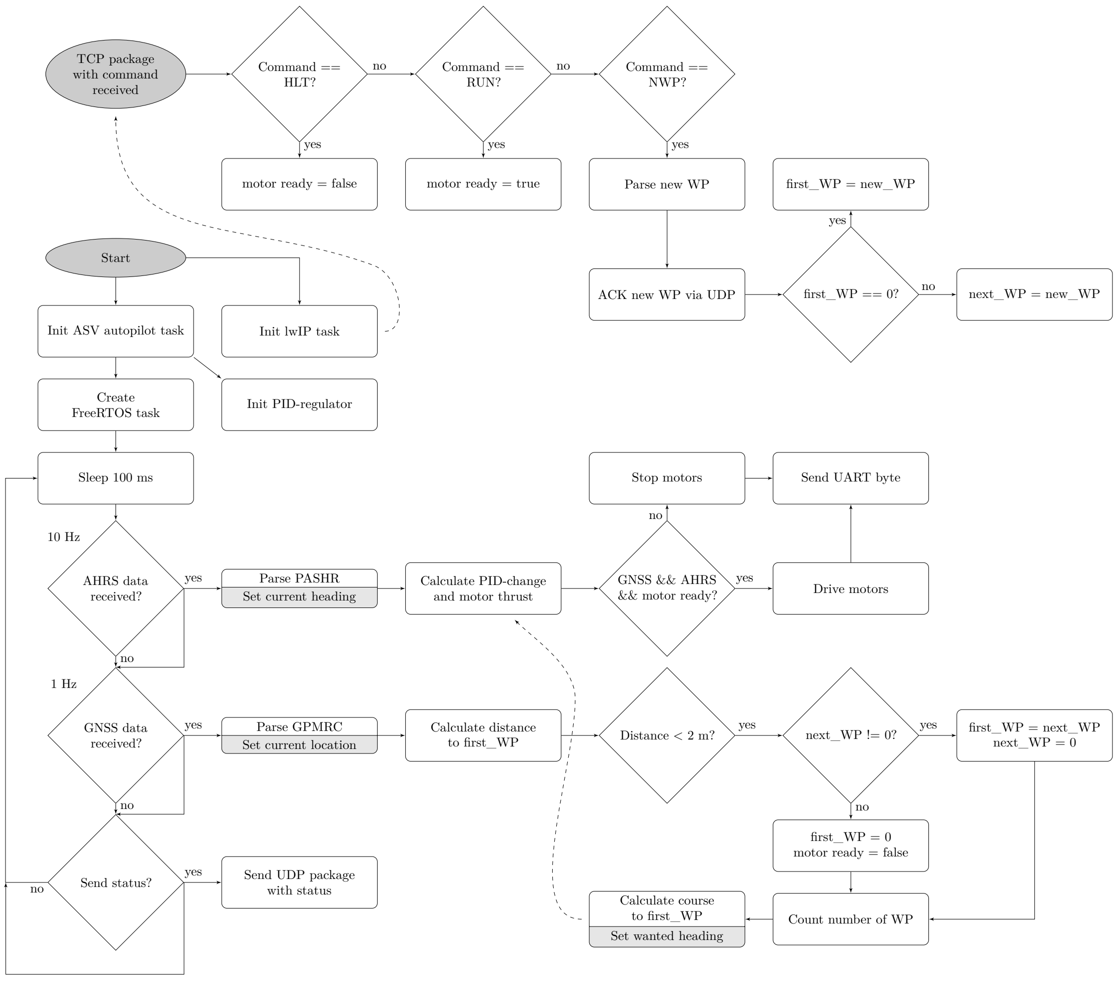


Figure 3.4: Block schematic of the autopilot algorithm and program flow. A callback function from lwIP starts the incoming TCP command parsing.

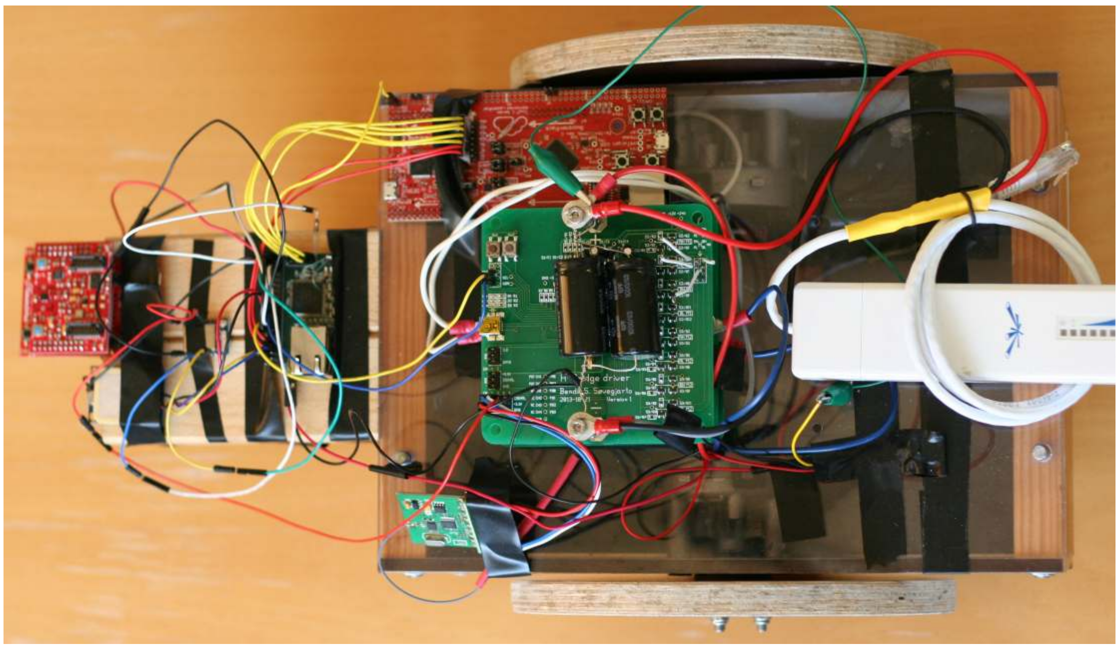


Figure 3.5: Prototype of the autopilot as multiple PCBs.

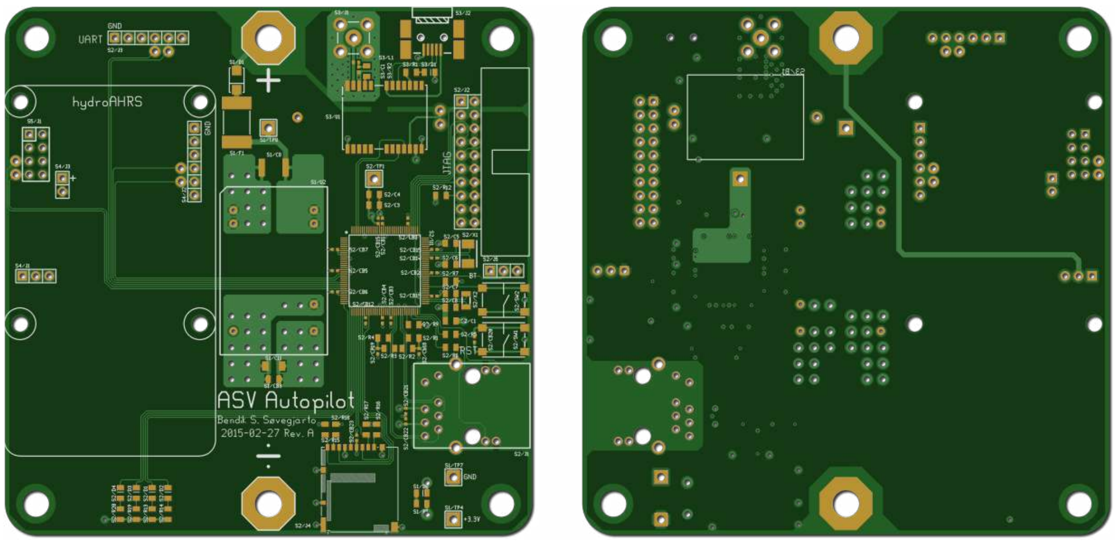


Figure 3.6: CGI of the autopilot PCB. Left image shows the top side, right image shows the bottom.

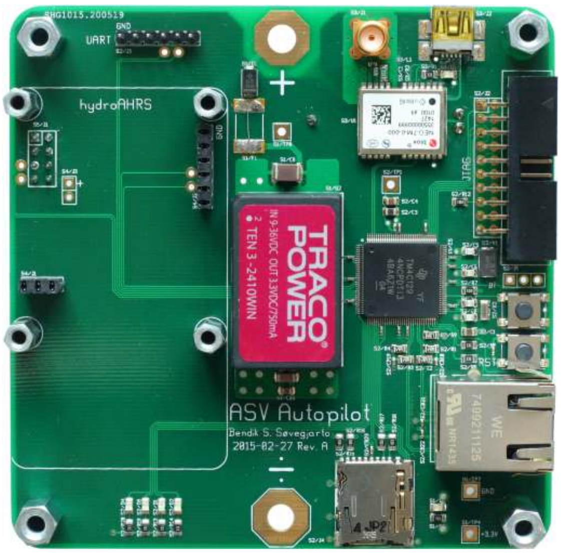


Figure 3.7: Photograph of assembled autopilot PCB.

3.3.6 Sub-system communication

All of the sub-systems communicate with the main MCU using UART. Inter-Integrated Circuit (I^2C) was considered, but UART was found to be sufficient since the sub-systems do not need to share a bus or talk directly to each other. UART is simpler to work with since there is no need for a complex driver. Each sub-system talks to a separate UART instance in the MCU. When an interrupt signals the arrival of a new byte from each sub-system, this byte is put into a ring buffer with a common interrupt handler (shared by all sub-systems). A command counter for each sub-system is increased when a newline is received. The newline terminates one command.

3.3.7 Operator communication

We need a way for the operator to control the autopilot. The radio control (RC)-communication used for the remote work platform is too simple and does not have the required bandwidth. Ethernet is a widely used and robust network technology that is fitting for communication with the autopilot.

A radio link consisting of two Ubiquity PicoStation M2 delivers a stable, encrypted and long range wireless connection between the operator and the ASV. This was tested in paper 1. These units were chosen because they are *a*) omni-directional; *b*) physically small; *c*) inexpensive; and *d*) manufactured by a well-known company specializing in wireless bridges.

TM4C129 has Ethernet capabilities embedded, but still requires quite a lot of software to utilize (e.g. a real-time operating system (RTOS)[4] and an Internet Protocol (IP)-stack). In the terms of the OSI model, the physical- (PHY) and data link-layer (MAC) is embedded in hardware, while the network- (IP) and transport-layer (UDP/TCP) must be implemented in software. A different communication device was therefore initially used.

A very cheap (~5 USD) Chinese chip named ESP8266 incorporates a wireless transceiver and UART peripherals, all in one package. The transceiver was used as a serial to WiFi bridge, connecting the autopilot to one of the PicoStations. This was, however, found to be quite unreliable.

[4]A RTOS is not required, but convenient if multiple tasks needs to be performed at the same time.

The firmware was therefore adapted to run FreeRTOS and lwIP to utilize the hardware Ethernet capabilities of TM4C129 to send status messages. The ESP8266 was thus abandoned. User Datagram Protocol (UDP) was chosen as the transport-layer for the status messages as the loss of some status packets are tolerated. Incoming commands are transported using TCP, since their arrival is crucial. The Media Access Control (MAC)-address (02:41:53:56:00:00) was set to a local administered address [37].

The PicoStation functions as a wireless Access Point (AP), but not as a router since we want the MCU and the client to reside within the same network. We use static IP-addresses, as the PicoStation cannot function as a Dynamic Host Configuration Protocol (DHCP)-server without being a router. The MCU has the static address 192.168.0.21, the PicoStation 192.168.0.20, and the client a different address in the 192.168.0/24 range[5].

Wireshark is a free[6] packet analyzer used to debug the Ethernet communication.

3.3.8 ASV Land

Many cycles of testing were needed while developing the autopilot. Since the autopilot relies on input from GNSS-satellites and needs room to move freely it is necessary to perform this testing outdoors. A land rover was therefore built to make the development of the autopilot easier. By using the land rover it is possible to test the autopilot frequently without having to travel to a lake. This saves time in planning, traveling and installation. It also makes it possible to mark the path (track) traveled on the ground. The land rover is shown in figure 3.8.

The land rover uses the same motor placement as a vessel on water. One motor on each side is connected to a wheel. A third wheel, which can rotate freely, is used for support. By controlling the rotational speed and direction of each of the side wheels, the land rover will steer and move in the same way as an ASV.

The land rover is built using plywood, wood, plastic laminated plates (Perstorp) and acrylic sheets. Two powerful motors were needed to drive the wheels. I visited a wrecking yard and bought two front wiper motors (8D1955113B by Bosch). Front wiper motors were chosen because of their high torque and easy availability.

The continues current draw of each of the motors without load was almost 5 A, while the peak inrush current was 8 A.

These motors have a RPM disk, making it possible to count their revolutions. Table 3.1 shows the revolutions per minute (RPM) for each of the motors. This was measured using an oscilloscope, with the disk connected through a pull-up resistor to a power source.

A maximum forward rotational speed of 110 RPM and a wheel diameter of 20 cm yields a maximum speed of about 4 km/h ($20 \times \pi \times 110 \frac{\text{cm}}{\text{minute}}$).

Table 3.1: Rotational speed for the two front motor wipers used in the land rover.

Motor	Backward [rpm]	Forward [rpm]
Motor 1	92	112
Motor 2	83	128

The wheels are mounted to the motor by pressing a mounting bracket on the wheel to a rilled shaft on the motor. This type of mounting limits the size of the wheel, as it cannot withstand large forces perpendicular to the rotation of the wheel.

3.3.9 ASV control panel

A web-based control panel has been developed to control and monitor the autopilot. This control panel shows the location of the ASV on a map, its current and wanted heading, its distance to the first waypoint and other status information. The control panel automatically sends new waypoints to the autopilot when a waypoint has been passed (number of waypoints

[5]The slash notation /24 means that 24 bits is used for the address space, and the remaining 8 bit is used for the host address space. 8 bit gives us 256 host addresses (0–255).

[6]as in free speech

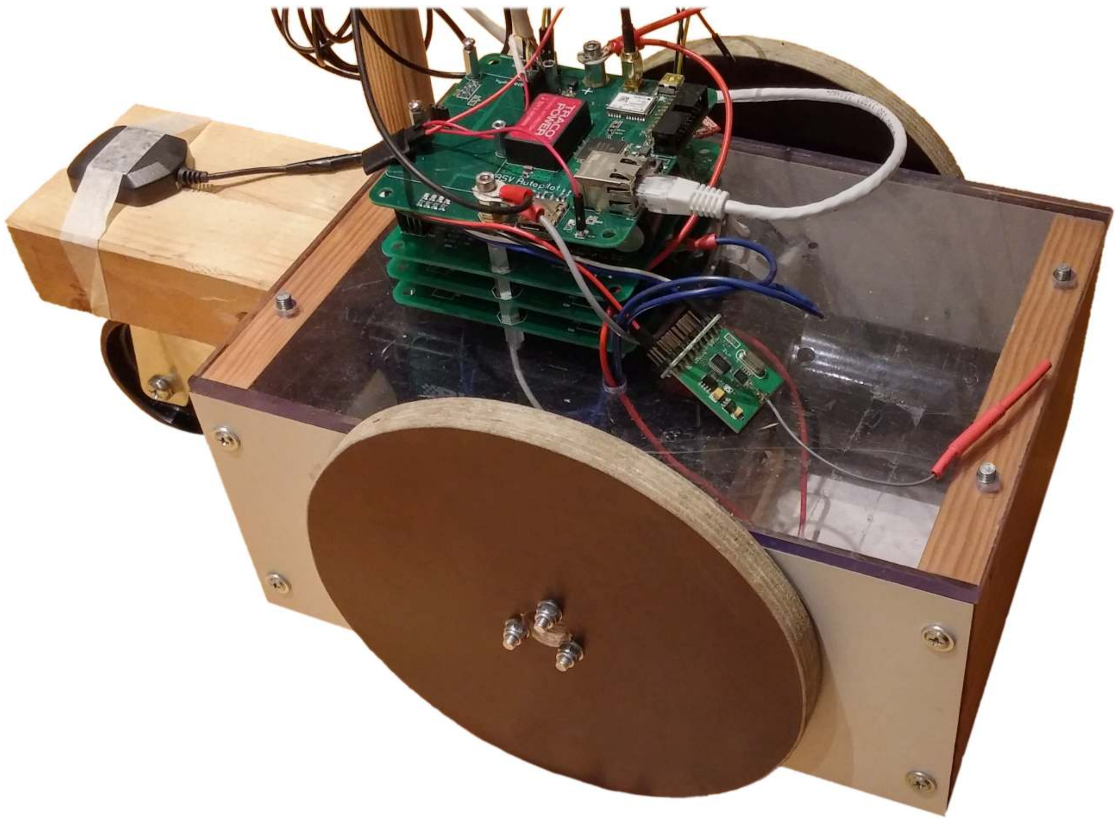


Figure 3.8: Photograph of land rover.

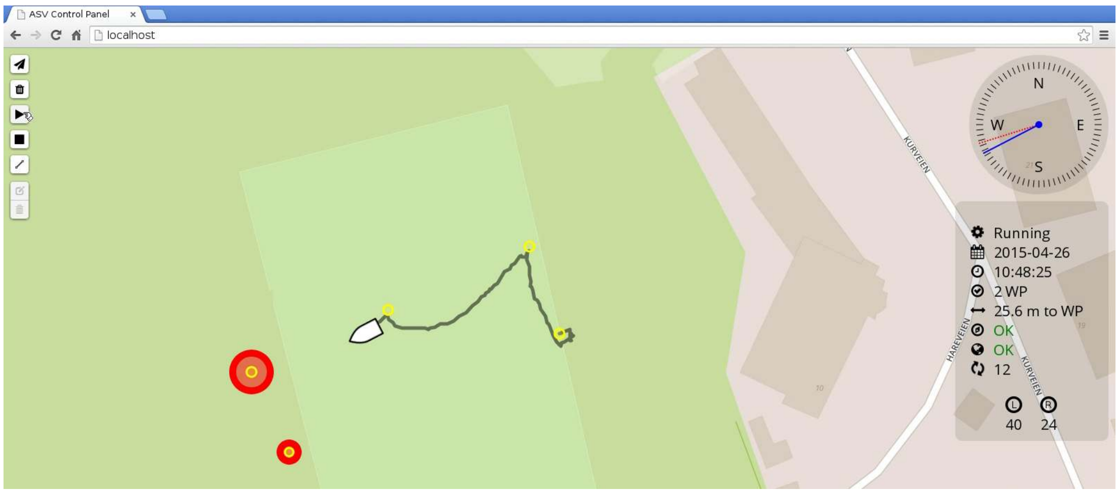


Figure 3.9: Web-based control panel for the ASV. The blue and red needle shows the current and wanted heading respectively. The red circles indicates the next waypoints.

reported by the autopilot is less than two). The control panel is shown in figure 3.9.

Node.js is used to relay the content of the UDP packages from the autopilot, over WebSockets (using Socket.IO) to the web-app. It also sends control messages over TCP back to the autopilot.

The code for the node.js server and control panel is listed in appendix N.

3.4 Testing and results

Three GNSS-receiver tests have been performed. One moving outdoor test, one long-term indoor stationary test and one short-term outdoor stationary test. The autopilot has been tested with the land rover.

3.4.1 Moving GNSS-receiver test

Keeping a GNSS-receiver stationary gives a good indication of its performance and this has been tested in the two consecutive test setups. We wanted, nevertheless, to make sure that there was no internal algorithm optimizing its position while being stationary. We would therefore like to test one of the GNSS-receivers (U-blox 6 PPP, with SBAS disabled) accuracy during movement. The best way to achieve this would be to have a good reference path for the GNSS-receiver to move along. A circular path created with a carousel was finally chosen because of the high accuracy and repeatability given by this setup. I built a large carousel using string, wood and aluminium bars as shown in figure 3.10. The antenna for the U-blox 6 PPP was placed at the end of the horizontal beam, 5.0 meter from the center.

The carousel was placed outside, and the data from the GNSS-receiver was recording to a Secure Digital (SD)-card using OpenLog[7]. We did four takes, moving the GNSS-receiver around multiple times at different speeds each time, as listed in table 3.2. The whole recording lasted 15 minutes and 50 seconds, and is showed in figure 3.11. The two last takes had few data points because of their short period.

The recording of the first and second take, consisting of 10 cycles with an average period of 22.5 and 17.4 seconds respectively, is shown in figure 3.12. The center of the circle was found by averaging all the points along the circle (of the second take). It is therefore not the exact center, but the best measure I had available. By assuming the Earth is flat over the area around the circle, I found that 0.0027 arc minutes latitude and 0.0054 arc minutes longitude corresponds to 5 meters from the center. The samples are scaled accordingly, to show the distance from the center.

The deviation from the circle was found by a Python program. The program translated all the values to decimal degrees and calculated the haversine distance for each one to the center. This distance should be 5 meters. The sample size is too small to calculate standard deviation. The error for the second take is shown in figure 3.13, along with the latitudinal and longitudinal position referenced to the center. The error is less than ± 1 m for this take. The first (and other) take had worse latitudinal accuracy, as shown in figure 3.11 and 3.12.

3.4.2 Long-term indoor stationary GNSS test

Three of the GNSS-receivers considered for use with the autopilot were tested over a period of 40 days in a stationary setup. The receivers considered were U-blox 6 PPP, U-blox NEO-6M and Trimble Copernicus II. The U-blox 6 PPP had SBAS disabled. The Trimble stopped functioning during the setup. All of them were fixed to one location during the whole experiment. They were placed indoors in a window sill for weather-protection, as shown in figure 3.14. We thought the view of the sky was adequate, and the receivers were able to find a minimum of four satellites at most times. The receivers output NMEA-messages at an interval of 1 Hz which were logged by a computer over Universal Serial Bus (USB). `Awk`[8]

[7]SparkFun OpenLog is an open source data logger. Available at `https://www.sparkfun.com/products/9530`
[8]`Awk` is an interpereted programming language commonly used for data extraction and text processing

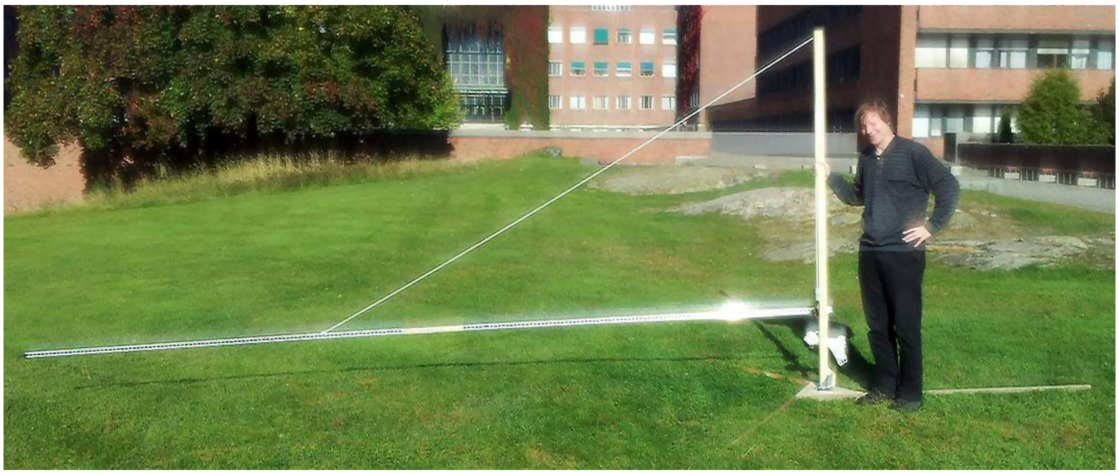


Figure 3.10: Carousel used in the moving GNSS-receiver test. The antenna of the GNSS-receiver is placed at the end of the horizontal beam. Joakim Myrland is standing next to it for scale.

Table 3.2: List of takes during carousel GNSS test.

Take #	From	To	# of per	Time [Sec]	Period $[\frac{\text{Sec}}{\#\text{ of per}}]$
1	13:17:53	13:21:38	10	225	22.5
2	13:22:36	13:25:30	10	174	17.4
3	13:26:37	13:28:07	10	90	9.0
4	13:29:33	13:30:07	5	34	6.8

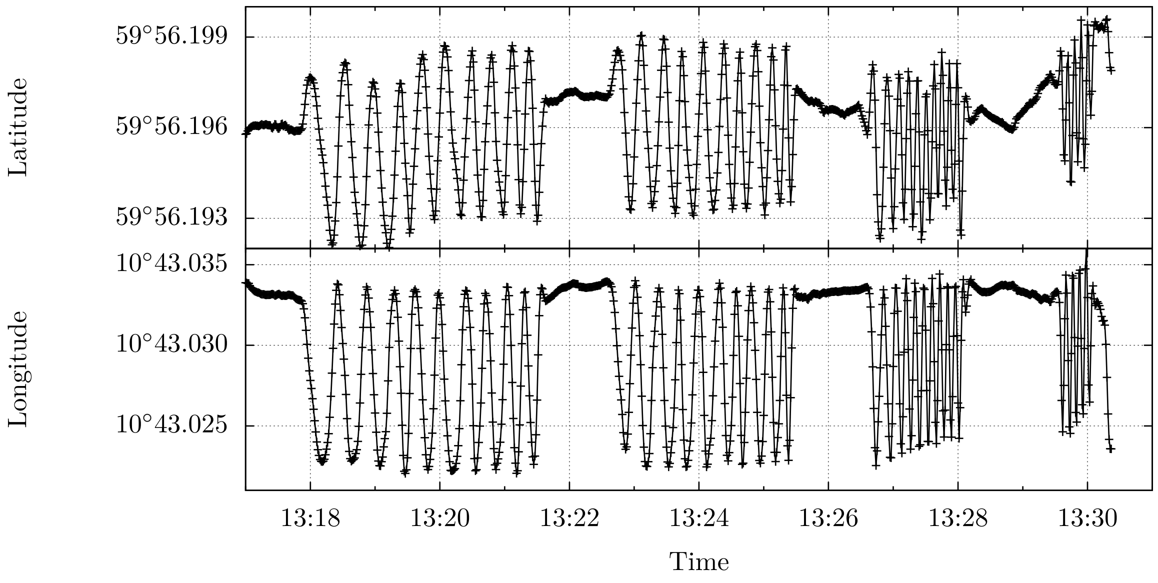
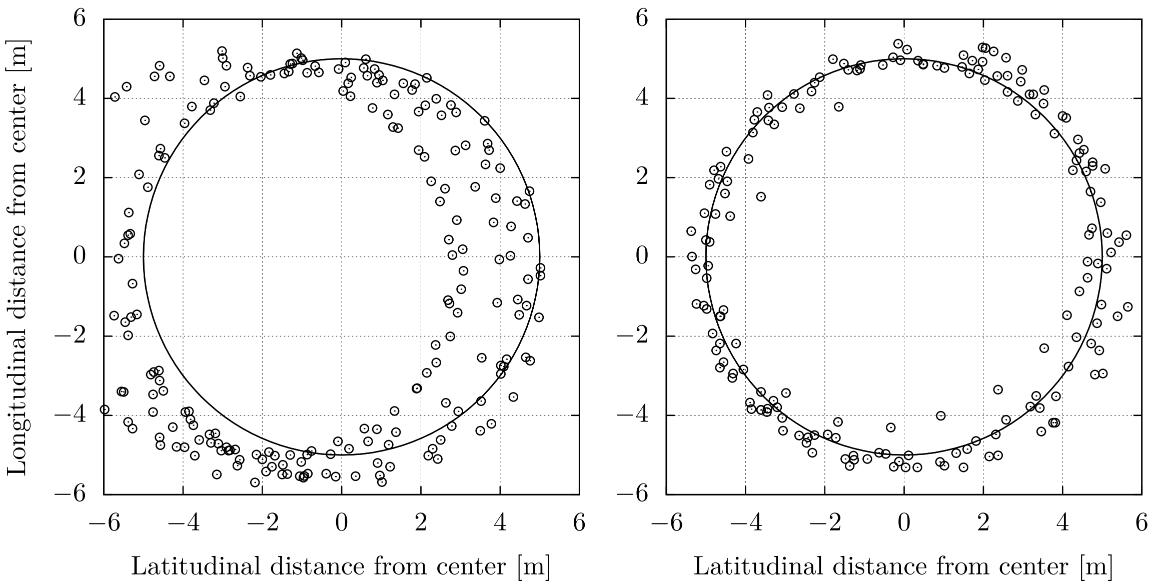


Figure 3.11: Raw latitude and longitude data of all takes in the GNSS carousel test.



(a) First take.

(b) Second take.

Figure 3.12: Plot of the distance from the center of the moving GNSS-circle to the mean, for the first and second take. The solid line marks the carousel.

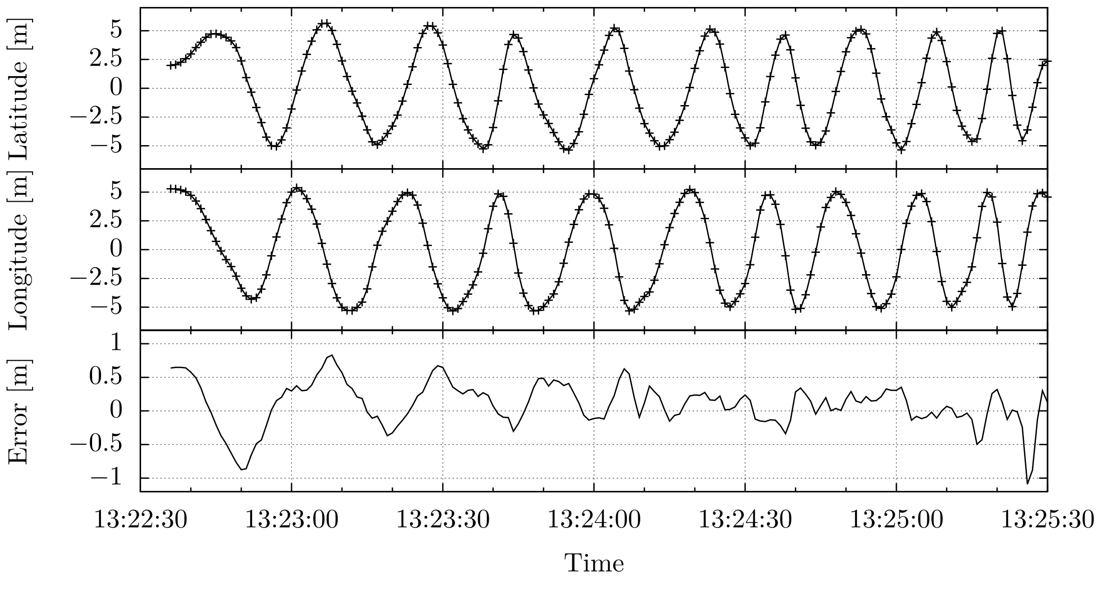


Figure 3.13: Data from the second GNSS carousel take. From the top: latitudinal positon relative to center, longitudinal positon relative to center and the error from the reference path.

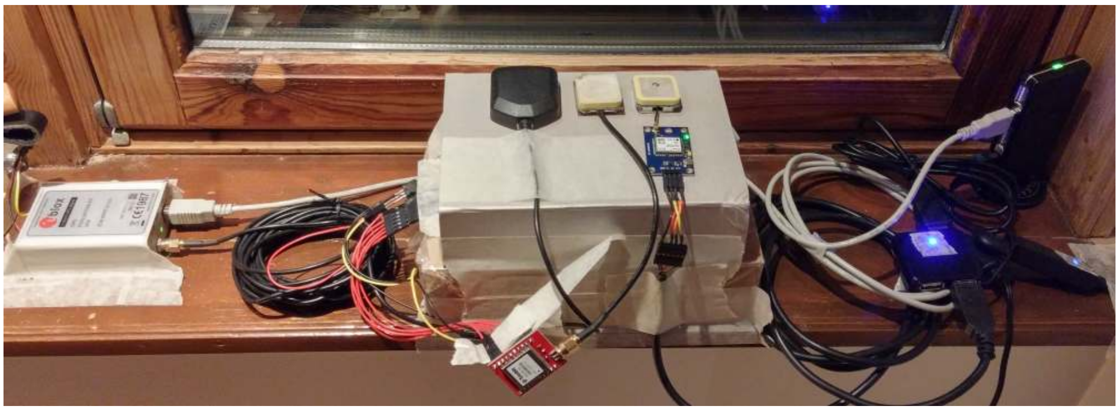


Figure 3.14: The indoor stationary GNSS-receiver test setup. The gray box to the left is the U-blox 6 PPP evaluation kit with the black antenna. The red PCB is the Trimble Copernicus II with the beige antenna in the middle. The blue PCB is the U-blox NEO-6M with the rightmost beige antenna.

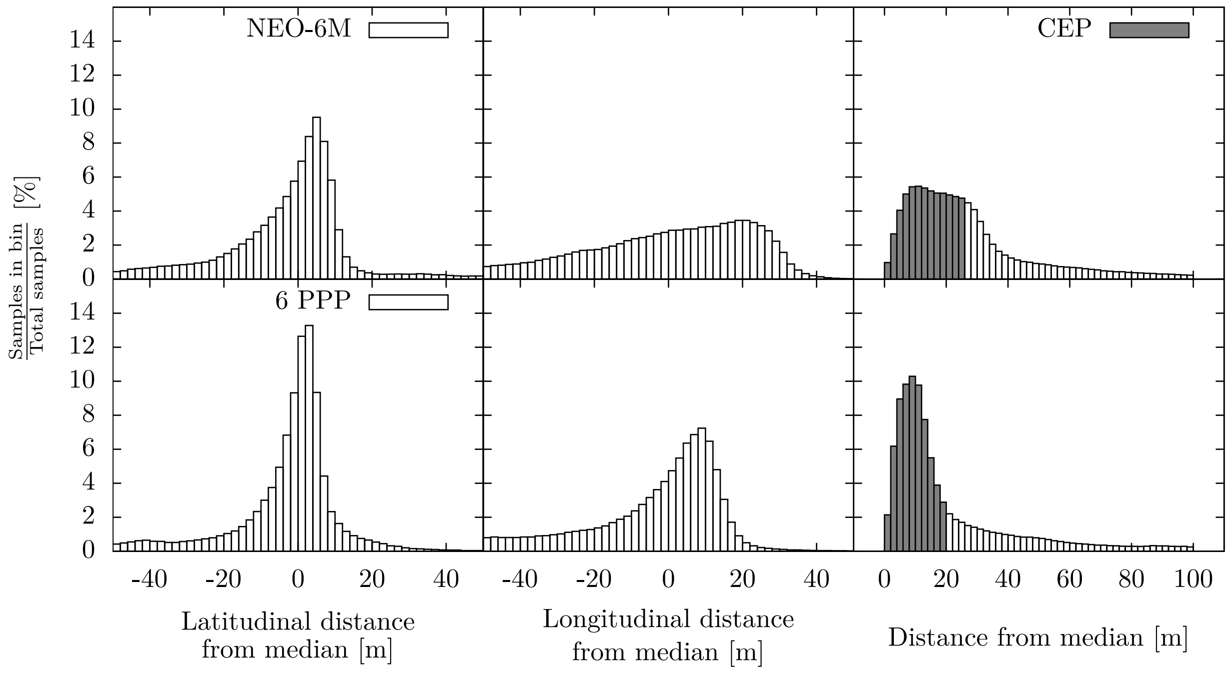


Figure 3.15: Histogram over the distance from the median coordinate, for the two GNSS-receivers used in the long time stationary indoor test. Bin-size is 2 m.

was used to extract the date, time, latitude and longitude position from the $GPRMC message and number of satellites in view and horizontal dilution of precision (HDOP) from the $GPGGA message, from 3.5 GB of NMEA 0183 (ASCII) data. The median location was calculated, and the distance from each sample was found using the haversine formula in Python. Figure 3.15 shows a histogram of the recorded data. U-blox 6 PPP had a CEP of 20.36 m, while U-blox NEO-6M had a CEP of 26.19 m in this test. Standard deviation was not calculated as the data did not have a normal distribution.

The results are discussed in section 3.5.1.

3.4.3 Outdoor stationary GNSS test

Due to the poor results from the indoor GNSS test, a new stationary test was set up outdoors. Five patch antennas were placed on the roof of a residential building, as shown in figure 3.16a. They were connected to five GNSS-receivers, as shown in figure 3.16b. Receiver 1 and 2 were U-blox NEO-M8N, receiver 3 and 4 were U-blox NEO-7M, and receiver 5 was the U-blox 6 PPP (with SBAS disabled) previously tested. Unit 1,2,3 and 4 were located on the autopilot PCB. Communication was initially done directly via USB to each GNSS-receiver, but USB communication for unit 1, 2, 3 and 4 stopped working after external power was applied to a USB-hub[9]. Data was therefore read from unit 1, 3 and 4 through the onboard MCU and sent to a computer via UART and FTDI (serial-to-USB) cables. Unit 2 was not used because not enough FTDI-cables were available.

The test was conducted over a period of 24 hours, in the same way as the indoor stationary test previously described.

The CEP, mean and standard deviation for each receivers latitudinal and longitudinal position was calculated, and is listed in table 3.3. A histogram of the data is shown in figure 3.17.

The results are discussed in section 3.5.2.

[9]The exact cause has not yet been investigated.

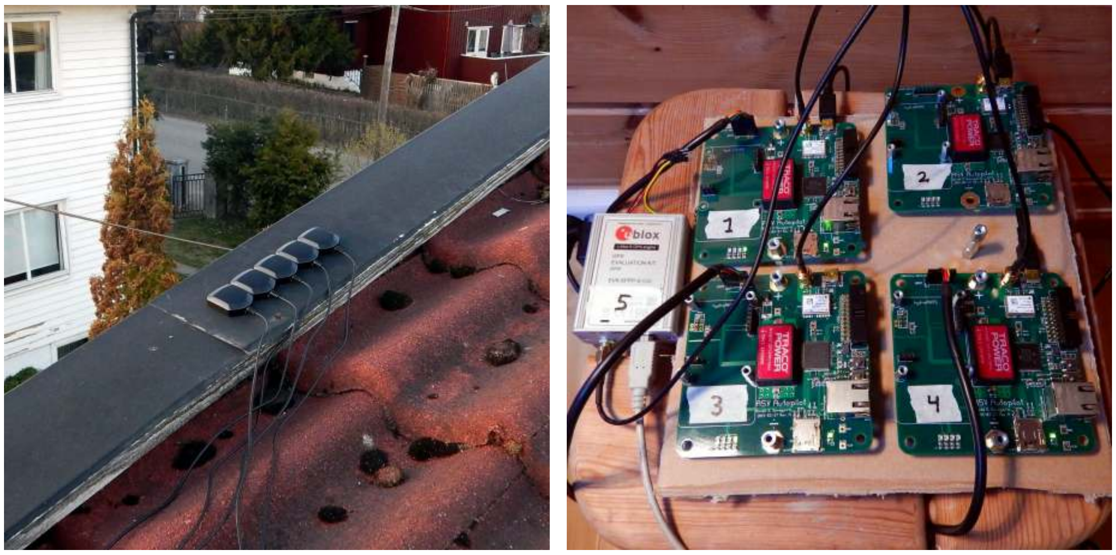
3.4.4 Autopilot test

The autopilot was tested on land using the land rover with a pre-defined route. I used the line-markings of a soccerfield with artifical turf to create a reference path for the autopilot to follow. I used a U-blox 7M GNSS-receiver to find the coordinates of five points on this path, by averaging the measured position for each point over a minimum of five minutes (shown in figure 3.20a). The coordinates were found six days before the test was conducted. These coordinates were used as waypoints for the autopilot.

Preliminary tests revealed that the compass behaved unreliable when mounted on top of the autopilot PCB. It suddenly changed its heading significantly within short time intervals. A LED indicating magnetic interference occasionally lit up as well. The compass was therefore mounted on top of a rod, to distance it from the magnetic fields from the motors and the GPS-antenna's magnet. The compass worked as expected and did no longer complain about interference after it was moved. The new setup is shown in figure 3.20b.

Another preliminary test showed oscillations when the land rover was trying to drive in a straight line. The autopilot tried to adjust its course, but overestimated the necessary thrust-difference needed to get the correct heading. The result was overshoot, and continues oscillations. The oscillations disappeared after the P-factor was decreased.

The land rover has different speeds on each motor, with the right wheel moving faster than the left wheel. This results in the land rover turning left, when the H-bridges is controlling them to move straight forward. This could have been fixed by implementing an adjustment factor for each individual motor in the steering controller of the H-bridge driver. We kept the asymmetrical behaviour, however, as it would give the autopilot a greater challenge and could be used as an simulation of the effect of external forces acting upon the ASV. For instance wind or a water current turning (but not drifting) the vessel.



(a) Five patch antennas mounted on a roof.

(b) Five GNSS-receivers. The four autopilot PCBs are shown to the right.

Figure 3.16: Outdoor stationary GNSS-receiver test setup.

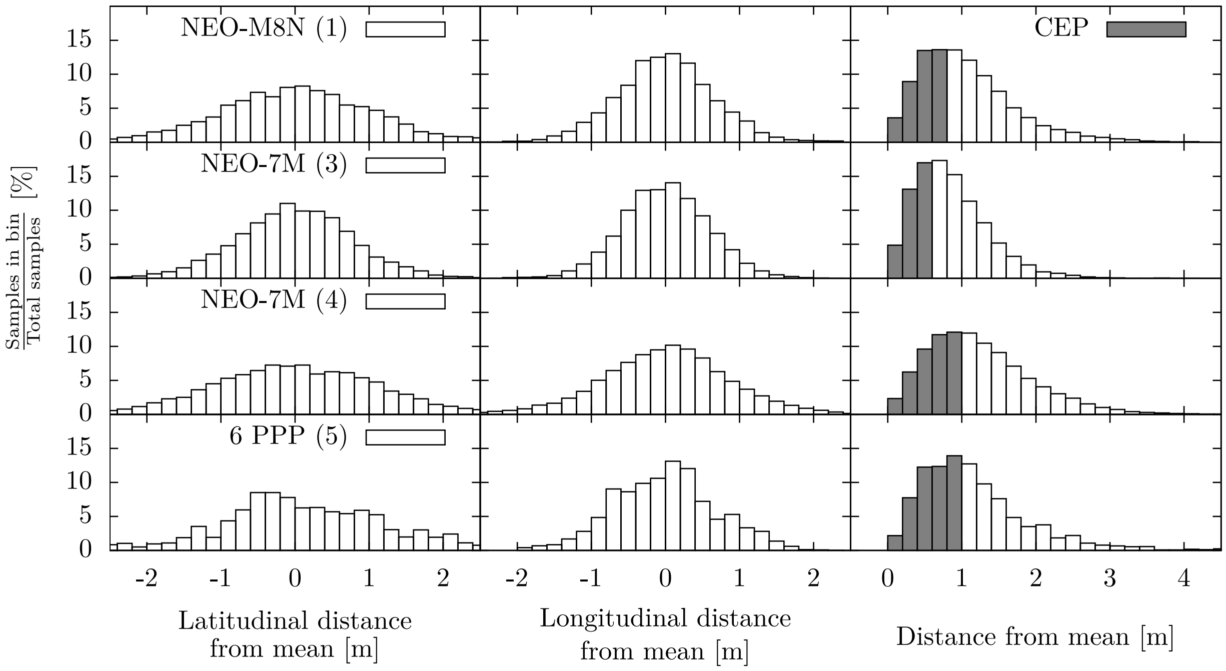


Figure 3.17: Histogram over the distance from the mean coordinate, for each of the four GNSS-receivers used in the outdoor stationary test. Bin-size is 20 cm.

Table 3.3: CEP, mean position and standard deviation (σ) for the outdoor GNSS test. * marks receivers with SBAS correction enabled.

		# of		Latitude		Longitude	
Unit	U-blox model	samples	CEP [m]	Mean [°]	σ [m]	Mean [°]	σ [m]
1	NEO-M8N*	60613	0.95	59.9573828	1.06	10.7873568	0.65
2	NEO-M8N*	0	–	–	–	–	–
3	NEO-7M*	86400	0.77	59.9573831	0.80	10.7873701	0.58
4	NEO-7M*	86400	1.13	59.9573792	1.12	10.7873627	0.88
5	6 PPP	86400	1.02	59.9573799	1.26	10.7873459	0.68

The land rover was placed at the start coordinate, with its heading pointing away from the first waypoint. A laptop running the ASV control panel was connected to the wireless LAN (WLAN) AP on the land rover. After a connection was established, the land rover was started by sending the first waypoints and activating its motors. The ASV control panel automatically sent new waypoints after the land rover passed each one. Waypoints were approved as passed when the GNSS-receiver reported a position within a 2 m perimeter from the waypoint's center. Status messages from the land rover was logged to file on the computer, as well as to an SD-card on the land rover (using the OpenLog).

To mark the path traveled, white stones were placed behind the support wheel of the land rover (located in the middle of the back). The distance from each stone to the reference line was later measured (together with its position along the line) as shown in figure 3.20c.The distance to the line was measured directly with a ruler and a tape measure. The position along the line was found by measuring the relative distance from the last stone. Since the relative measurements results in an accumulation of errors, each line position was finally scaled according to the total length of the line. A total of 243 stones were placed and measured.

The reference path and the path traveled (from the measured stones and GPS) are shown in figure 3.18. The wanted heading (calculated by the autopilot) and the current heading (measured by the compass) for the whole transect is shown in figure 3.19.

The results are discussed in section 3.5.3.

3.5 Discussion

The following paragraphs discusses test results, problems encountered and possible enhancements for further development of the autopilot.

3.5.1 Long-term indoor GNSS test

A median value was calculated instead of a mean because of outliers and a skewed distribution. The median gave a value closer to the mode. The latitudinal accuracy was better than the longitudinal. The window sill (and wall) the GNSS-receivers were placed in is almost parallel to the meridian, and this may explain the bad longitudinal accuracy.

It is clearly visible that the indoor test was inadequate, as the calculated CEP was greater than 20 meters. We found both very low accuracy and precision, much worse than the previous (moving) GNSS test. The long-term indoor GNSS test was thus incapable of giving an impression of the performance of the GNSS-receivers tested.

3.5.2 Outdoor GNSS test

The outdoor GNSS test was only conducted in 24 hours, but still provides an indication of the performance of the GNSS receivers. Unit

[10]`gpsd` is a free (as in free speech) GPS service daemon. `http://www.catb.org/gpsd/`

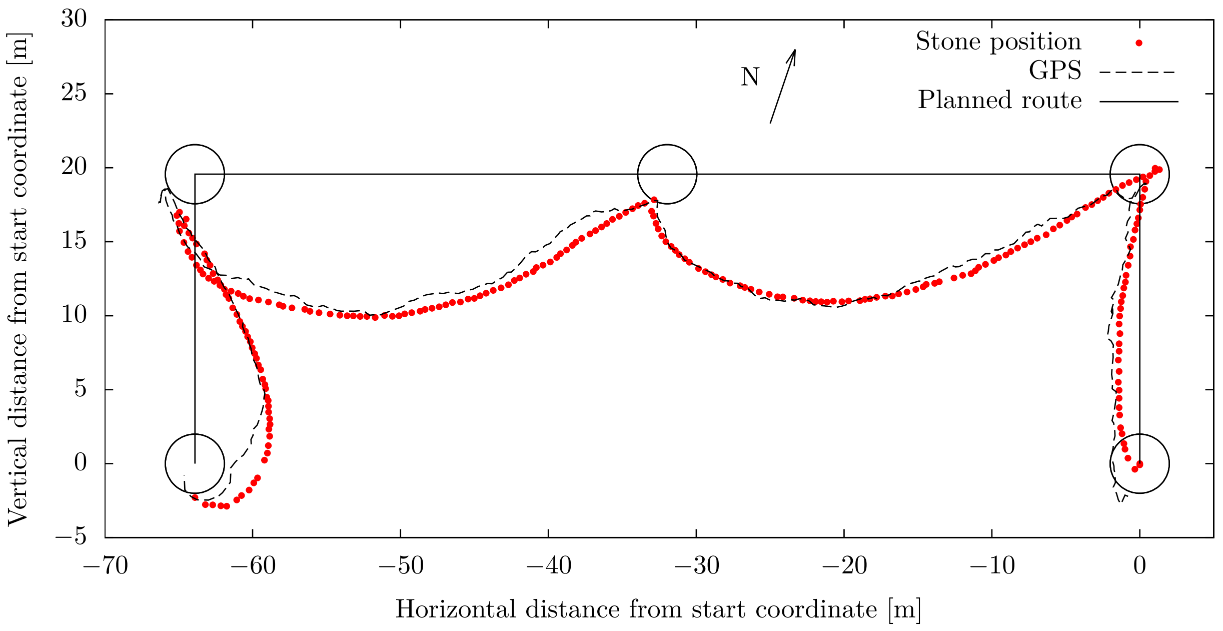


Figure 3.18: Reference path and path traveled (from measured stones and GPS) for the land rover autopilot test. The black circles have a radius of 2 meter, and shows the perimeter of the waypoints.

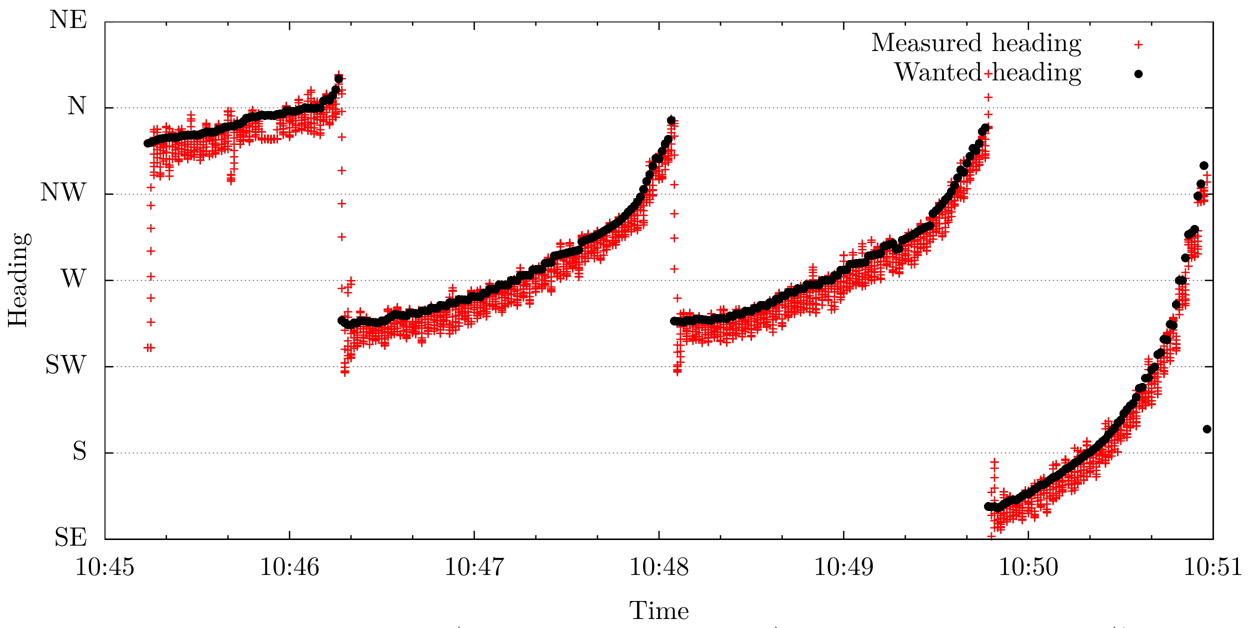
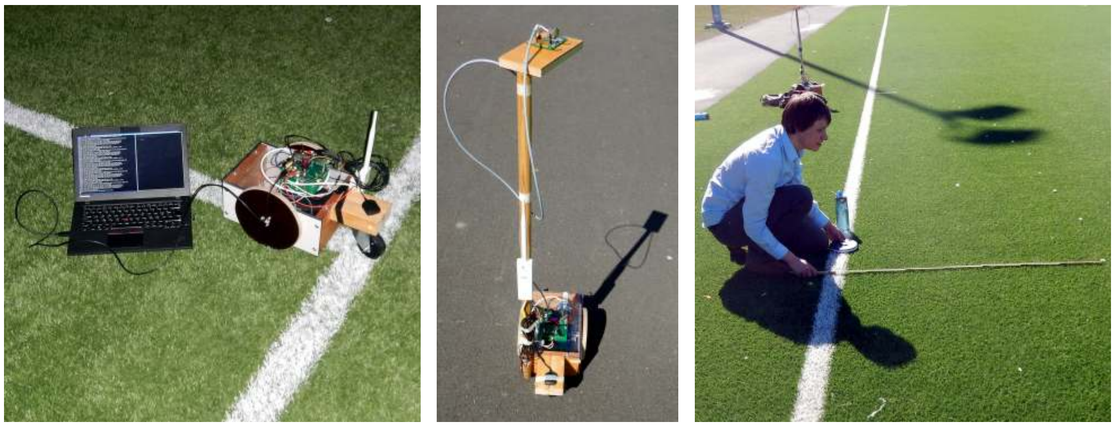


Figure 3.19: Wanted heading (calculated by the autopilot) and the measured heading (from the compass) for the land rover autopilot test.



(a) Setup used to find the coordinates for the waypoints. (b) Land rover with elevated compass. (c) Measurement setup to find stone positions.

Figure 3.20: Photographs of the autopilot test setup.

1 had almost 30 % of its samples missing or invalid. This unit had earlier been automatically reconfigured by `gpsd`[10] and outputted different NMEA messages than the others. Some samples may have been corrupted when passing through the MCU. The longitudinal accuracy was consistently better than the latitudinal, and the overall accuracy was similar for all the receivers. SBAS was disabled (by error) on the U-blox 6 PPP, which makes a comparison between PPP and non-PPP units difficult. The accuracy found in the test was close to our expectations given by the manufacturer's specifications. A CEP of about 1 meter is sufficient for the autopilot.

3.5.3 Autopilot test

The autopilot managed to pass through all its waypoints (figure 3.18) and stopped when it arrived at the last. It did, however, drive between each waypoint in large arcs. We believe some of this behaviour is caused by the asymmetry in motor thrust, but the main source is the compass reporting wrong headings. This can be seen by the difference in the northbound (first and second waypoint) and the southbound (last and second to last waypoint) arc. The compass used should therefore be tested and investigated closer. The error may be solved by using a lookup-table between the real and measured heading from the compass. This look-up table can be generated by driving in a full circle at a constant speed while the measured heading angles from the compass is recorded. Another way to create the look-up table would be to rotate the compass to certain known (relative) angles and interpolate the values in between.

The maximum deviation from the reference path was encountered halfway into the third arc, with a distance of ~9.69 m to the reference line. The positions reported by the GPS corresponds to the measured stone positions, for the most part of the route.

The behaviour of the PID-regulator can be seen in figure 3.19. The measured heading is continuously adjusted to follow the wanted heading. The asymmetry of the motor thrust (that causes the rover to turn left) can also be seen, as the measured heading is almost always located below the wanted heading.

3.5.4 Communication enhancements

The status messages is currently sent to the broadcast address 255.255.255.255. This address should be changed to an address in the

IPv4 local multicast scope (239.255.0.0/16) [38, p. 2] to avoid spamming the whole network.

The new waypoint acknowledgement sent back from the autopilot to the control panel should be transferred using TCP instead of UDP, since the UDP package may never arrive.

3.5.5 Implement WebSocket in the MCU

The best way to communicate with the autopilot would be to use the Ethernet capabilities of the TM4C129 and implement a Hypertext Transfer Protocol (HTTP)- and WebSocket-server in the firmware. WebSockets enables full-duplex realtime communication directly between the front end software (webpage) and the MCU. The webpage could be served by the MCU, or preloaded on a device. By implementing everything in the MCU, very little is required by the operating device, just a modern web browser which supports WebSockets and other HyperText Markup Language (HTML) 5 components. Any laptop, tablet or even smart phone may thus be used to directly control the vessel, without the need of a separate computer.

3.5.6 PID-regulator

We are at the moment only using the P-factor in the PID-regulator, so it is currently only a P-regulator.

3.5.7 Approximations

The bearing changes while traveling between two points on a sphere (unless a rhumb line is followed), but we only use the initial bearing (forward azimuth) when calculating the course. We believe this is a valid approximation as the bearing is recalculated every second (after each GNSS-sample).

We assume a spherical Earth with a radius of 6373 km.

3.5.8 Autopilot and motor driver communication bug

A bug between the autopilot and the motor driver results in the motor driver stopping the motors for a couple of seconds at seemingly random occasions. This happens if the motor drivers thinks the autopilot has stopped sending commands. The bug is visible 45 seconds into the heading-recordings of figure 3.19, as the heading-measurements were constant when the land-rover stopped temporarily. The cause of the bug has not yet been found.

3.5.9 ARM toolchain

The change to a new MCU platform was quite time-consuming, since a whole new toolchain was used. We faced challenges with the linker-script (mapping of the memory areas), dynamic memory allocation, and the floating-point unit (FPU). Stack-overflows and hardfaults were frequent. The final software solution has, however, proved to be stable and easy to expand. The use of a RTOS makes it very easy to add new tasks in seamless concurrency.

3.5.10 True north

The use of the magnetic north as reference instead of the true north causes an error when navigating. We believe this can be tolerated for the short distances (< 1 km) we operate with, and with the small magnetic declination of only ~2.8 ° in Oslo [39, 40].

3.5.11 GNSS ready flag

The autopilot is currently halted if no GNSS fix is available. A more strict check could be used instead, e.g. by requiring a minimum number of visible satellites. Alternatively let the autopilot run, but warn the user that the accuracy is low.

3.5.12 Alternative software

OpenCPN is a free[11] chartplotter and navigation software project. It may be sensible to use

[11] as in free speech

or intergrate OpenCPN in the autopilot, as this is a mature and widely used software solution for maritime navigation.

Other free autopilot-systems exists, e.g. Paparazzi [41]. Although this is made for aircrafts, some of the code base may be applicable for a maritime autopilot.

3.5.13 Future work

A more advanced autopilot should be developed before the system is usable for hydroacoustic work. It could for instance learn from its past history and use it to predict the future. A Kalman filter would be suitable for a new autopilot.

We had a requirement of constant speed. If minimum survey time is wanted, the thrust could be adjusted depending on the distance to the next waypoint.

Another improvement would be to use more than one waypoint to calculate the course. For instance by cutting corners and even out the track.

Additional sensors can be included in the control system to improve and extend the functionality of the ASV. For instance a RADAR for obstacle avoidance, an echosounder for depth measurements, water- and wind movement sensors for improved navigation and a (IP) camera for supervision.

3.6 Summary

We have developed a primitive autopilot for hydroacoustic work using an inexpensive GNSS-receiver and our self-developed compass. The autopilot is running on a 32-bit ARM Cortex M4 MCU, and talks with its sub-systems (GNSS, compass, motor control system) over UART. A web-based control panel has been developed to observe and control the vessel. Status messages are transferred as UDP-packages over Ethernet to a nearby computer, which relays the data using WebSockets to the web-based control panel. Control commands are sent back to the autopilot using TCP.

Different positioning solutions have been considered, but the use of a single GNSS-receiver was chosen. Multiple GNSS-receivers have been tested, and a suitable low cost GNSS-receiver has been found.

A land rover has been built as a development platform for the autopilot. The autopilot has been tested on the land rover, and was able to follow waypoints, but with significant deviations from the planned route.

Summary

This thesis has presented the development of an Automatic Survey Vessel (ASV). The goal was to create a low cost control unit which could be placed in an arbitrarily selected full-sized boat and be used to make hydroacoustic surveying on lakes more efficient. The work was presented through three papers.

Paper 1, «Construction of a remote controlled work platform for hydroacoustic work», covered the development of a motor control system. Two attempts on creating a (H-bridge based) motor control system for brushed DC-motors have been completed and is described in this paper. The final attempt has proved itself capable of controlling electric outboard engines powerful enough to propel a full-sized vessel. The motor control system has remote control functionality which allows an operator to steer the vessel from a distance. Tests conducted with a vessel carrying a scientific echosounder (with wireless transmission of data) has shown that the system can be used as a remote controlled work platform for hydroacoustic work. It was observed that switching-noise generated by the motor control system may corrupt hydroacoustic recordings, and this must be addressed in the future.

Paper 2, «Construction of an Attitude and Heading Reference System (AHRS)», covered the development of a compass needed for navigation. Three attempts on creating an AHRS have been completed and is described in this paper. The tilt-calculations from the second attempt has been tested in controlled- and fieldwork-environments and fulfilled our criteria of repeatability and accuracy. Preliminary testing of the heading-calculations (compass) from the third attempt showed promising results.

Paper 3, «Construction of a primitive autopilot for hydroacoustic work», used the motor control system from paper 1 and the compass from paper 2 to create a simple autopilot able to steer a full-sized boat. A primitive proof-of-concept autopilot capable of following dynamically generated waypoints has been developed and is described in this paper. The autopilot has been tested on a land rover with partial success. The land rover was able to pass through all waypoints, but with significant deviations from the planned route.

We have accomplished to lay the foundation for an Automatic Survey Vessel (ASV) platform, with a functional motor control system, a compass and a simple autopilot. A more advanced autopilot can reuse all the hardware developed, or just the motor control system and compass. Future work includes *a*) suppressing the motor-switching noise; *b*) testing and improving the reliability of the compass; *c*) incorporating a (local) DGPS solution; *d*) improving the autopilot, e.g. with the use of a Kalman filter; *e*) extending the ASV with multiple sensors (cameras, echosounders, RADAR, water- and wind movement); *f*) including obstacle avoidance and other safety requirements; and *g*) finally making the ASV autonomous with ongoing route planning.

Closing

I have learned a lot, and had much fun while working on my master thesis. 13 printed circuit boards (PCBs) have been designed in association with this project, where 8 have been directly involved (and described in this thesis). This includes designs used to get familiar with some of the components, but excluding revisions with only minor adjustments. A total of 52 PCBs have been assembled.

I have designed hardware, written firmware in C, computer software in Python, JavaScript (node.js), Awk and Bash, as well as web-based front-ends in JavaScript/HTML/CSS.

Appendix A

Installation of ASV control panel

The following guide will install and setup the ASV control software on a fresh install of Debian Wheezy where sudo access is available. Installation of Debian is not covered in this guide, but can in short steps be done as described in step 0.

1. Add a new package repository (Debian Backports) to get access to newer software, e.g. `nodejs`.
   ```
   echo "deb http://http.debian.net/debian wheezy-backports main" | sudo tee -append
   /etc/apt/sources.list
   ```
2. Update available packages
   ```
   sudo apt-get update
   ```
3. Install `nodejs` (with a symlink for `node`)
   ```
   sudo apt-get -t wheezy-backports install nodejs-legacy
   ```
4. Install `git` and `chromium`, as well as necessary software to install `npm`
   ```
   sudo apt-get install git chromium curl build-essential
   ```
5. Install and update `npm`
   ```
   curl -L -insecure https://www.npmjs.org/install.sh | sudo sh
   sudo npm install -g npm
   ```
6. Install ASV control panel
   ```
   mkdir git && cd git
   git clone https://github.com/epsiro/ASV_control_panel
   cd ASV_control_panel/
   ```
7. Install `node-static` and `socket.io`
   ```
   npm install node-static
   npm install socket.io
   ```
8. Start the ASV control panel server
   ```
   nodejs server.js
   ```
9. Open the URL localhost with chromium. Alternatively the ASV control panel may be accessed from another computer/tablet/smartphone by opening the IP address of the machine were the ASV control panel was installed. E.g. 192.168.4.101 . The IP address may be found using `ifconfig`.

0. Install Debian Wheezy

    (a) Download the correct image (e.g. amd64 live) from debian.org

    (b) Use dd on a linux machine to transfer the image to a usb stick. Be very careful not to overwrite the wrong device! e.g. `dd if=debian.img of=/dev/sdX bs=4M; sync` where X is your device.

    (c) Insert the USB stick into a computer and choose it as the prefered boot device by configuring the BIOS.

    (d) Follow the on-screen instructions for installation.

    (e) Add yourself to the sudo group to get SuperUser privileges by editing the file /etc/group as root and log off and on again.
    `su`
    then
    `nano /etc/group`

    (f) Configure network

Appendix B

PCB production

One PCB is designed in Zuken CadStar. All other PCBs have been designed in gEDA pcb[1]. My self-developed footprint editor[2] was used to create many of the footprints. Very little proprietary software has been used in my master thesis.

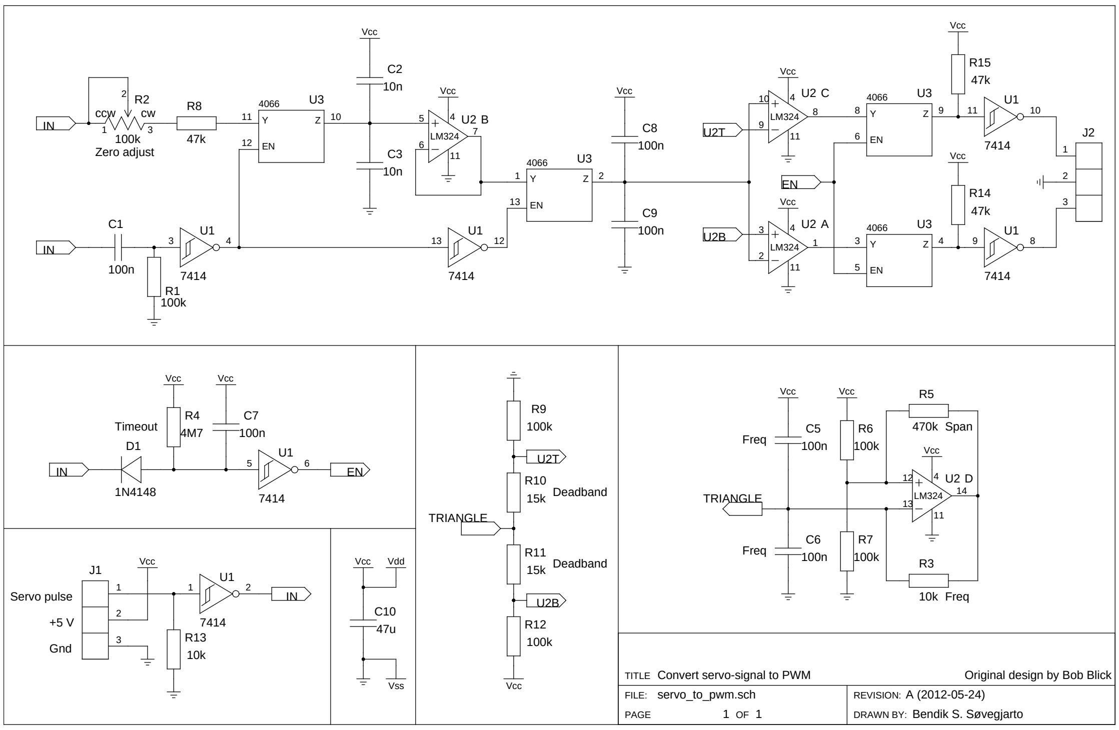
The large PCBs (involved with the H-bridge) is mainly manufactured by Seeed Technology Inc. a company located in Shenzhen, China. The smaller PCBs is manufactured by OshPark, located in the United States of America (US). This is due to their different price models. The autopilot PCB was manufactured through the electronic laboratory at UiO's connection with Elprint.

[1]`http://pcb.geda-project.org/`

[2]Code available at `https://github.com/epsiro/pcb-footprint-editor`. Demo available at `http://pcb.zone` in 2015.

Appendix C

Analog steering controller schematics



Appendix D

MCU code for servo to PWM signal converter

Code for ATtiny85 proof of concept servo to PWM signal converter, as described in section 1.3.1. The code is licensed under the MIT license (listing D.1).

Listing D.1: MIT license

code/attiny_servo_to_pwm.c

```c
#define F_CPU 1000000

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#define SPEED PB1
#define DIR PB3
#define DD_SPEED DDB1
#define DD_DIR DDB3
#define SERVO_IN PCINT4

void init() {
    /****** For the out signal ******/
    /* SPEED and DIR as output */
    DDRB = 1<<SPEED | 1<<DIR;

    /* Mode: Fast PWM */
    TCCR0A = 1<<WGM01 | 1<<WGM00;
    TCCR0B = 0<<WGM02;

    /* No prescaling of counter clock */
    TCCR0B |= 0<<CS02 | 0<<CS01 | 1<<CS00;

    /* OC0B on Compare Match, sets OC0B at BOTTOM */
    TCCR0A |= 1<<COM0B1 | 0<<COM0B0;

    /****** For the in signal ******/
    /* Globally enable interrupts */
    sei();

    /* Enable pin change interrupt */
    GIMSK = 1<<PCIE;

    /* Pin change interrupt is enabled on SERVO_IN */
    PCMSK = 1<<SERVO_IN;

    /* Start a normal counter1 with prescaler /4 */
    TCCR1 = 1<<CS11 | 1<<CS10;
}

void main() {
    init();

    for (;;) {
    }
}

ISR(PCINT0_vect) {
    /* We have a rising edge */
    if (PINB & 1<<PB4) {
        /* Reset counter1 */
        TCNT1 = 0;

    /* We have a falling edge */
    } else {
        if (TCNT1 >= 123 && TCNT1 <= 131) {
                PORTB &= ~1<<DIR;
                OCR0B = 0;
        } else {
            /* Motor forward */
            if (TCNT1 & 0x80) {
                PORTB |= 1<<DIR;

                /* Set the PWM duty cycle */
                /* (TCNT1 - 0x80)*2 */
                OCR0B = (TCNT1 & ~0x80)<<1;

            /* Motor backward */
            } else {
                PORTB &= ~1<<DIR;

                /* Set the PWM duty cycle */
                OCR0B = (0x80 - TCNT1)<<1;
            }

        }
    }
}
```

Appendix E

MCU code for first digital steering controller

Code for ATxmega128A1 digital steering controller, as described in section 1.3.1. The code is licensed under the MIT license (listing D.1).

code/HIP4081_controller/main.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdint.h>

#define F_CPU 8000000UL

#define PWM_FREQ 1000L
#define PER (F_CPU/PWM_FREQ)

#define sec_to_tic(sec) sec*F_CPU
#define CENTER_LEN 1.5e-3
#define DEAD_BAND  0.02e-3

#define LEFT_STICK 0x00
#define RIGHT_STICK 0x01

static int16_t min_pulse_width[2];
static int16_t max_pulse_width[2];

static int8_t left_stick;
static int8_t right_stick;

static int16_t left_motor_fwd;
static int16_t left_motor_bck;
static int16_t right_motor_fwd;
static int16_t right_motor_bck;

void
clock_init() {

    /* RC2M->PLL, 4x multiplier */
    OSC.PLLCTRL = OSC_PLLSRC_RC2M_gc | 0x4;

    /* Start PLL */
    OSC.CTRL |= OSC_PLLEN_bm;

    /* Wait for PLL to become stable */
    while (!(OSC.STATUS & OSC_PLLRDY_bm));

    /* Disable ccp "Configuration Change Protection" */
    CCP = CCP_IOREG_gc;

    /* Use PLL output as the system clock */
    CLK.CTRL = CLK_SCLKSEL_PLL_gc;
}

void
servo_init() {

    /* Configure PC0 and PC1 for input */
    PORTC.DIRCLR = PIN0_bm;
    PORTC.DIRCLR = PIN1_bm;

    /* Interrupt on both edges. */
    PORTC.PIN0CTRL = PORT_ISC_BOTHEDGES_gc;
    PORTC.PIN1CTRL = PORT_ISC_BOTHEDGES_gc;

    /* Select PC0 as input to event channel 0,
     *     and PC1 as input to event channel 1. */
    EVSYS.CH0MUX = EVSYS_CHMUX_PORTC_PIN0_gc;
    EVSYS.CH1MUX = EVSYS_CHMUX_PORTC_PIN1_gc;

    /* Pulse width capture, using event channel 0 and 1 */
    TCC1.CTRLD = TC_EVSEL_CH0_gc | TC_EVACT_PW_gc;

    /* Enable Input "Capture or Compare" channel A and B. */
    TCC1.CTRLB |= TC0_CCAEN_bm | TC0_CCBEN_bm;

    /* Set the period of the counter. Quote from datasheet:
     *
     * If the Period register value is set lower than 0x8000, the polarity of
     * the I/O pin edge will be stored in the Most Significant Bit (MSB) of the
     * Capture register after a Capture. If the MSB of the Capture register is
     * zero, a falling edge generated the Capture. If the MSB is one, a rising
     * edge generated the Capture.
     */
    TCC1_PER = 0x7fff;

    /* Start timer by selecting a clock source. No prescaling. */
    TCC1.CTRLA = TC_CLKSEL_DIV1_gc;

    /* Enable CCA and CCB interrupt */
    TCC1.INTCTRLB = TC_CCAINTLVL_LO_gc | TC_CCBINTLVL_LO_gc;
}

void
pwm_init() {

    /* Enable the PWM outputs, PE0, PE1, PE2 and PE3 */
    PORTE.DIR = 0b00001111;

    /* Set the TC period */
    TCE0_PER = PER;

    /* Configure the TC for single slope mode. */
    TCE0.CTRLB = TC_WGMODE_SS_gc;

    /* Enable Compare channel A, B, C and D. */
    TCE0.CTRLB |= TC0_CCAEN_bm | TC0_CCBEN_bm | TC0_CCCEN_bm | TC0_CCDEN_bm;

    /* Start timer by selecting a clock source. No prescaler. */
    TCE0.CTRLA = TC_CLKSEL_DIV1_gc;
}

void
int_init() {

    /* Enable low level interrupt */
    PMIC.CTRL |= PMIC_LOLVLEN_bm;

    /* Enable global interrupt */
    sei();
}
```

```

int
main() {

    // TODO: Save this in eeprom and add calibration button
    min_pulse_width[LEFT_STICK] = -3431L;
    max_pulse_width[LEFT_STICK] =  3379L;

    min_pulse_width[RIGHT_STICK] = -3423L;
    max_pulse_width[RIGHT_STICK] =  3385L;

    clock_init();
    servo_init();
    pwm_init();
    int_init();

    for(;;) {
    }
}

void
calibrate_stick(int16_t * pulse_width, int8_t stick) {

    if (*pulse_width < min_pulse_width[stick]) {
        min_pulse_width[stick] = *pulse_width;
    }

    if (*pulse_width > max_pulse_width[stick]) {
        max_pulse_width[stick] = *pulse_width;
    }
}

void
drive_motors() {

    /* Left and right stick is in middle position (within dead band) =>
     * Stand still */
    if (left_stick == 0 && right_stick == 0) {
        left_motor_fwd = 0;
        left_motor_bck = 0;
        right_motor_fwd = 0;
        right_motor_bck = 0;

    /* Turn on the spot */
    } else if (left_stick == 0) {
        left_motor_fwd = right_stick;
        left_motor_bck = -right_stick;
        right_motor_fwd = -right_stick;
        right_motor_bck = right_stick;

    /* Drive forwards */
    } else if (left_stick > 0) {
        left_motor_fwd = left_stick + right_stick;
        left_motor_bck = 0;
        right_motor_fwd = left_stick - right_stick;
        right_motor_bck = 0;

    /* Drive backwards */
    } else if (left_stick < 0) {
        left_motor_fwd = 0;
        left_motor_bck = -left_stick - right_stick;
        right_motor_fwd = 0;
        right_motor_bck = -left_stick + right_stick;
    }

    /* Remove overflows */
    if (left_motor_fwd < 0) left_motor_fwd = 0;
    if (left_motor_bck < 0) left_motor_bck = 0;
    if (right_motor_fwd < 0) right_motor_fwd = 0;
    if (right_motor_bck < 0) right_motor_bck = 0;

    if (left_motor_fwd > 127) left_motor_fwd = 127;
    if (left_motor_bck > 127) left_motor_bck = 127;
```

```c
    if (right_motor_fwd > 127) right_motor_fwd = 127;
    if (right_motor_bck > 127) right_motor_bck = 127;

    /* Set PWM registers */
    TCE0_CCABUF = (left_motor_fwd*PER)/127L;
    TCE0_CCBBUF = (left_motor_bck*PER)/127L;
    TCE0_CCCBUF = (right_motor_fwd*PER)/127L;
    TCE0_CCDBUF = (right_motor_bck*PER)/127L;
}

int8_t
normalize_stick_position(int16_t * pulse_width, uint8_t stick) {

    int16_t stick_position;

    /* Center around 1.5 ms */
    *pulse_width -= sec_to_tic(CENTER_LEN);

    /* Dead band */
    if (abs(*pulse_width) <= sec_to_tic(DEAD_BAND)) {
        stick_position = 0;

    /* Scale to -127..127 */
    } else if (*pulse_width >= 0) {
        stick_position = (*pulse_width * 127L)/max_pulse_width[stick];
    } else {
        stick_position = -(*pulse_width * 127L)/min_pulse_width[stick];
    }

    /* 125..127 -> 127 */
    if (stick_position >= 125) {
        stick_position = 127;
    }

    return (int8_t) stick_position;
}

ISR(TCC1_CCA_vect) {
    left_stick = normalize_stick_position(&TCC1_CCA, LEFT_STICK);
    drive_motors();
}

ISR(TCC1_CCB_vect) {
    right_stick = normalize_stick_position(&TCC1_CCB, RIGHT_STICK);
}
```

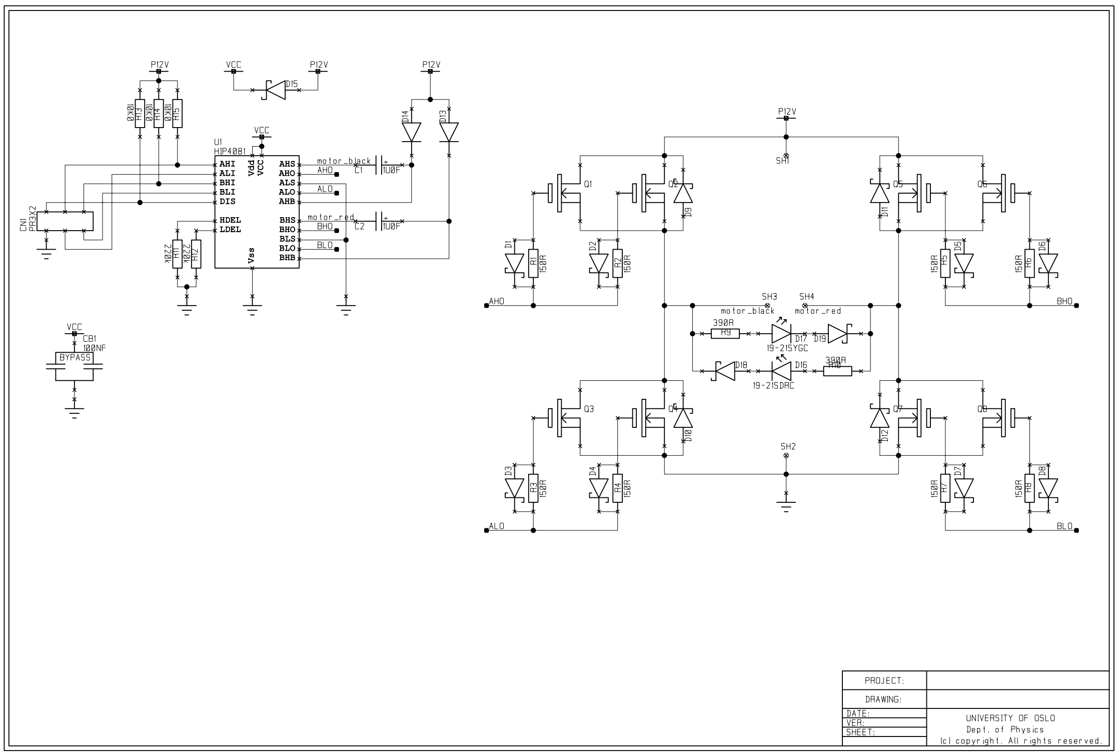
code/HIP4081_controller/Makefile

```makefile
DEVICE = atxmega128a1
AVRDUDE = avrdude -p x128a1 -c jtagmkII -P usb
AVARICE = avarice --mkII -j usb --xmega :4242

CC = avr-gcc
CFLAGS = -g -W -Wall -O2 -std=c99 -mmcu=$(DEVICE)

OBJECTS = main.o

debug: flash
	sleep 2
	$(AVARICE) &

flash: all
	$(AVRDUDE) -U flash:w:main.hex:i

all:	main.hex

clean:
	rm -f *.hex *.lst *.o *.bin

$(OBJECTS): | depend

main.bin: $(OBJECTS)
```

```
	$(CC) $(CFLAGS) -o main.bin $(OBJECTS)

main.hex: main.bin
	avr-objcopy -j .text -j .data -O ihex main.bin main.hex
	avr-size --totals *.o

read:
	$(AVRDUDE) -U eeprom:r:eeprom.dat:r
	hd eeprom.dat

depend:
	@$(CC) -MM $(ALL_CFLAGS) *.c | sed 's/$$/ Makefile/'
```

Appendix F

First H-bridge schematics



Appendix G

MCU code for ASV H-bridge controller

Code for ATxmega32A4U digital steering and H-bridge controller, as described in section 1.3.2. A block schematic of the program flow is shown in figure 1.18. The code is also available at `https://github.com/epsiro/ASV_H-bridge_controller` and is licensed under GNU General Public License (GPL) version 2 or later.

code/ASV_H–bridge_controller/main.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

#define BAUD_D   9600UL
#define BSEL_D(baud)  ((F_CPU / (16 * baud)) - 1)

#define PWM_FREQ 1000L
#define PER (F_CPU/(2*PWM_FREQ))

#define STATE_FREQ 100L
#define STATE_PER ((F_CPU/8)/STATE_FREQ)

#define RC_FREQ 50L

#define NUMBER_OF_RC_CMD_TO_AVERAGE 8

#define sec_to_tic(sec) sec*F_CPU
#define CENTER_LEN 1.5e-3
#define DEAD_BAND  0.04e-3

#define ON PER
#define OFF 0
#define DELAY (4*PER)/100L

#define LEFT_STICK 0x00
#define RIGHT_STICK 0x01

#define LEFT_MOTOR 0x00
#define RIGHT_MOTOR 0x01

#define FORWARD 1
#define BACKWARD -1

#define AH 0x00
#define AL 0x01
```

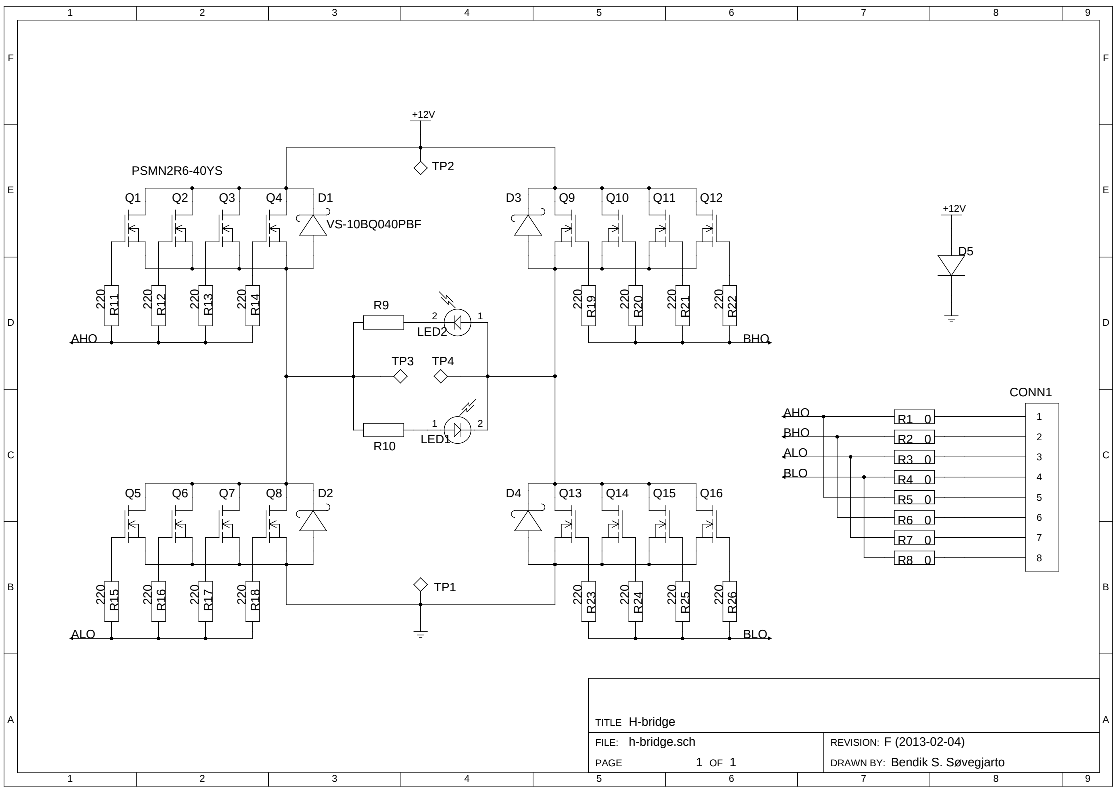
```
#define BH 0x02
#define BL 0x03

#define TRUE 0x01
#define FALSE 0x00

static volatile uint16_t* drive[4][2] = {
    {&TCC0_CCABUF, &TCC0_CCCBUF},
    {           0,            0},
    {&TCC0_CCBBUF, &TCC0_CCDBUF},
    {           0,            0}};

static int16_t min_pulse_width[2];
static int16_t max_pulse_width[2];

static int8_t left_stick = 0;
static int8_t right_stick = 0;
static int8_t average_motor_thrust[2] = {0, 0};

static volatile int8_t uart_motor_thrust[2] = {0, 0};

static volatile uint32_t number_of_rc_commands = 0;
static volatile uint32_t number_of_runs_without_rc_command = 0;
static volatile uint8_t rc_receiver_ready = FALSE;

static volatile uint32_t number_of_uart_commands = 0;
static volatile uint32_t number_of_runs_without_uart_command = 0;
static volatile uint8_t uart_receiver_ready = FALSE;

void
clock_init() {

    /* RC2M->PLL, 4x multiplier */
    OSC.PLLCTRL = OSC_PLLSRC_RC2M_gc | 0x4;

    /* Start PLL */
    OSC.CTRL |= OSC_PLLEN_bm;

    /* Wait for PLL to become stable */
    while (!(OSC.STATUS & OSC_PLLRDY_bm));

    /* Disable ccp "Configuration Change Protection" */
    CCP = CCP_IOREG_gc;

    /* Use PLL output as the system clock */
    CLK.CTRL = CLK_SCLKSEL_PLL_gc;
}

void
rc_init() {

    /* Configure PB0 and PB1 for input */
    PORTB.DIRCLR = PIN0_bm;
    PORTB.DIRCLR = PIN1_bm;

    /* Interrupt on both edges. */
    PORTB.PIN0CTRL = PORT_ISC_BOTHEDGES_gc;
    PORTB.PIN1CTRL = PORT_ISC_BOTHEDGES_gc;

    /* Select PB0 as input to event channel 0,
     *    and PB1 as input to event channel 1. */
    EVSYS.CH0MUX = EVSYS_CHMUX_PORTB_PIN0_gc;
    EVSYS.CH1MUX = EVSYS_CHMUX_PORTB_PIN1_gc;

    /* Pulse width capture, using event channel 0 and 1 */
    TCC1.CTRLD = TC_EVSEL_CH0_gc | TC_EVACT_PW_gc;

    /* Enable Input "Capture or Compare" channel A and B. */
    TCC1.CTRLB |= TC0_CCAEN_bm | TC0_CCBEN_bm;

    /* Set the period of the counter. Quote from datasheet:
     *
     * If the Period register value is set lower than 0x8000, the polarity of
     * the I/O pin edge will be stored in the Most Significant Bit (MSB) of the
     * Capture register after a Capture. If the MSB of the Capture register is
     * zero, a falling edge generated the Capture. If the MSB is one, a rising
     * edge generated the Capture.
     */
    TCC1_PER = 0x7fff;

    /* Start timer by selecting a clock source. No prescaling. */
    TCC1.CTRLA = TC_CLKSEL_DIV1_gc;

    /* Enable CCA and CCB interrupt */
    TCC1.INTCTRLB = TC_CCAINTLVL_LO_gc | TC_CCBINTLVL_LO_gc;
}

void
fsm_init() {

    /* Set the period of the counter */
    TCD0_PER = STATE_PER;

    /* Start timer by selecting a clock source. /8 prescaling. */
    TCD0.CTRLA = TC_CLKSEL_DIV8_gc;

    /* Enable overflow interrupt */
    TCD0.INTCTRLA = TC_OVFINTLVL_HI_gc;
}

void
pwm_init() {

    /* Enable the PWM outputs for the left and right motor
     * Left:  PC0, PC1, PC2 and PC3
     * Right: PC4, PC5, PC6 and PC7
     */
    PORTC.DIR = 0xFF;

    /* Set the TC period */
    TCC0_PER = PER;

    /* Configure the TC for dual slope mode. */
    TCC0.CTRLB = TC_WGMODE_DS_T_gc;

    /* Enable Compare channel A, B, C and D. */
    TCC0.CTRLB |= TC0_CCAEN_bm | TC0_CCBEN_bm | TC0_CCCEN_bm | TC0_CCDEN_bm;

    /* Start timer by selecting a clock source. No prescaler. */
    TCC0.CTRLA = TC_CLKSEL_DIV1_gc;

    /* Enable AwEx Dead Time Insertion */
    AWEXC.CTRL |= AWEX_DTICCAEN_bm | AWEX_DTICCBEN_bm
                | AWEX_DTICCCEN_bm | AWEX_DTICCDEN_bm;

    /* Set the dead time (both high and low)
     * in number of CPU cycles
     * 240 ticks = 30us */
    AWEXC.DTBOTH = 240;

    /* Override all pins */
    AWEXC.OUTOVEN = 0xff;

    /* Invert all pins */
    PORTC.PIN0CTRL |= PORT_INVEN_bm;
    PORTC.PIN1CTRL |= PORT_INVEN_bm;
    PORTC.PIN2CTRL |= PORT_INVEN_bm;
    PORTC.PIN3CTRL |= PORT_INVEN_bm;
    PORTC.PIN4CTRL |= PORT_INVEN_bm;
    PORTC.PIN5CTRL |= PORT_INVEN_bm;
    PORTC.PIN6CTRL |= PORT_INVEN_bm;
    PORTC.PIN7CTRL |= PORT_INVEN_bm;

    /* Make sure we have a safe start */
    *drive[AH][LEFT_MOTOR] = OFF;
    *drive[BH][LEFT_MOTOR] = OFF;
    *drive[AH][RIGHT_MOTOR] = OFF;
    *drive[BH][RIGHT_MOTOR] = OFF;

}

void
int_init() {

    /* Enable high and low level interrupt */
    PMIC.CTRL |= PMIC_HILVLEN_bm;
    PMIC.CTRL |= PMIC_LOLVLEN_bm;

    /* Enable global interrupt */
    sei();
}

void
button_init() {

    /* Configure PE2 and PE3 for input */
    PORTE.DIRCLR = PIN2_bm;
    PORTE.DIRCLR = PIN3_bm;
}

void
led_init() {

    /* Set Green LED-pin PD4 as output */
    PORTD.DIRSET = PIN4_bm;

    /* Set Red LED-pin PD5 as output */
    PORTD.DIRSET = PIN5_bm;

    /* Invert the LED pins, so the LEDs light when we set them high */
    PORTD.PIN4CTRL |= PORT_INVEN_bm;
    PORTD.PIN5CTRL |= PORT_INVEN_bm;
}

void
uart_init(void) {

    /* Set PD3 (TX) as output */
    PORTD.DIRSET = PIN3_bm;

    /* Set the baud rate */
    uint16_t bsel = BSEL_D(BAUD_D);
    USARTD0.BAUDCTRLB = (bsel >> 8) & 0xFF;
    USARTD0.BAUDCTRLA = (uint8_t) bsel;

    /* Enable low level UART receive interrupt */
    USARTD0.CTRLA = USART_RXCINTLVL_LO_gc;

    /* Enable the UART transmitter and receiver */
    USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

    /* Asynchronous USART, no parity, 1 stop bit, 8 data bits */
    USARTD0.CTRLC = USART_CHSIZE0_bm | USART_CHSIZE1_bm;
}

int
main() {

    // FIXME: Save this in eeprom and add calibration button
    min_pulse_width[LEFT_STICK] = -3431L;
    max_pulse_width[LEFT_STICK] =  3379L;

    min_pulse_width[RIGHT_STICK] = -3423L;
    max_pulse_width[RIGHT_STICK] =  3385L;

    clock_init();
    rc_init();
    fsm_init();
    pwm_init();
    int_init();
    button_init();
    led_init();
    uart_init();

    //PORTD.OUTSET = PIN4_bm;
    //PORTD.OUTSET = PIN5_bm;

    for(;;) {
    }
}

void
calibrate_stick(int16_t * pulse_width, int8_t stick) {

    if (*pulse_width < min_pulse_width[stick]) {
        min_pulse_width[stick] = *pulse_width;
    }

    if (*pulse_width > max_pulse_width[stick]) {
        max_pulse_width[stick] = *pulse_width;
    }
}

void
single_sticks() {

    drive_motor(LEFT_MOTOR, left_stick);
    drive_motor(RIGHT_MOTOR, right_stick);
}

void
combo_sticks() {

    int16_t motor_thrust_left;
    int16_t motor_thrust_right;

    if (left_stick == 0 && right_stick != 0) {

        /* Turn on the spot */
        motor_thrust_left  =  right_stick;
        motor_thrust_right = -right_stick;

    } else {

        /* Drive */
        motor_thrust_left  = left_stick + right_stick;
        motor_thrust_right = left_stick - right_stick;
    }

    /* Remove overflows */
    if (motor_thrust_left  >  127) motor_thrust_left  =  127;
    if (motor_thrust_left  < -127) motor_thrust_left  = -127;
    if (motor_thrust_right >  127) motor_thrust_right =  127;
    if (motor_thrust_right < -127) motor_thrust_right = -127;

    drive_motor_lp(LEFT_MOTOR,  (int8_t) motor_thrust_left);
    drive_motor_lp(RIGHT_MOTOR, (int8_t) motor_thrust_right);
}

void
drive_motor_lp(int8_t motor, int8_t motor_thrust) {

    /* Calculate moving average */
    average_motor_thrust[motor] = ( (int32_t)
            (NUMBER_OF_RC_CMD_TO_AVERAGE - 1)*average_motor_thrust[motor]
            + motor_thrust)/NUMBER_OF_RC_CMD_TO_AVERAGE;

    drive_motor(motor, average_motor_thrust[motor]);
}

void
drive_motor(int8_t motor, int8_t motor_thrust) {

    /* Scale the pwm-value */
    uint16_t pwm = (abs(motor_thrust)*PER)/127L;

    if (motor_thrust > 0) {

        /* Drive forward */
        *drive[AH][motor] = OFF;
        *drive[BH][motor] = pwm;

    } else {

        /* Drive backward */
        *drive[AH][motor] = pwm;
        *drive[BH][motor] = OFF;
    }
}

int8_t
normalize_stick_position(int16_t * pulse_width, uint8_t stick) {

    int16_t stick_position;

    /* Center around 1.5 ms */
    *pulse_width -= sec_to_tic(CENTER_LEN);

    /* Dead band */
    if (abs(*pulse_width) <= sec_to_tic(DEAD_BAND)) {
        stick_position = 0;

    /* Scale to -127..127 */
    } else if (*pulse_width >= 0) {

        stick_position = (*pulse_width * 127L)/max_pulse_width[stick];

    } else {

        stick_position = -(*pulse_width * 127L)/min_pulse_width[stick];
    }

    /* 125..127 -> 127 */
    if (stick_position >= 125) {
        stick_position = 127;
    }

    return (int8_t) stick_position;
}

ISR(TCC1_CCA_vect) {

    number_of_runs_without_rc_command = 0;


    if (rc_receiver_ready == TRUE) {
    left_stick = normalize_stick_position(&TCC1_CCA, LEFT_STICK);
    } else {

        /* Skip the first interrupts since the receiver is not stable then */
        if (++number_of_rc_commands >= 1000*RC_FREQ*2) {
            rc_receiver_ready = TRUE;
        }
    }
}

ISR(TCC1_CCB_vect) {
    if (rc_receiver_ready == TRUE) {
        right_stick = normalize_stick_position(&TCC1_CCB, RIGHT_STICK);
    }
}

ISR(TCD0_OVF_vect) {

    /* Turn off both LEDs, so they will light only when they should */
    PORTD.OUTCLR = PIN4_bm;
    PORTD.OUTCLR = PIN5_bm;

    /* Timeout if we do not get any rc commands.
     * Triggered after waiting for two missing commands */
    if (++number_of_runs_without_rc_command >= 2*(STATE_FREQ/RC_FREQ)) {
        rc_receiver_ready = FALSE;
        number_of_rc_commands = 0;
    }

    /* Timeout if we do not get any uart commands. */
    if (++number_of_runs_without_uart_command >= 20) {
        uart_receiver_ready = FALSE;
        number_of_uart_commands = 0;
    }

    if (rc_receiver_ready == TRUE) {
    //if (PORTE.IN & PIN2_bm)
        //single_sticks();
    //else

        /* Set red LED to show that we are using commands from RC receiver */
        PORTD.OUTSET = PIN5_bm;

        combo_sticks();

    } else if (uart_receiver_ready == TRUE) {

        /* Set green LED to show that we are using commands from UART */
        PORTD.OUTSET = PIN4_bm;

        drive_motor(LEFT_MOTOR, uart_motor_thrust[LEFT_MOTOR]);
        drive_motor(RIGHT_MOTOR, uart_motor_thrust[RIGHT_MOTOR]);

    } else {

        drive_motor(LEFT_MOTOR, 0);
        drive_motor(RIGHT_MOTOR, 0);
    }
}

ISR(USARTD0_RXC_vect) {

    number_of_runs_without_uart_command = 0;

    uint8_t motor;
    int8_t direction;

    uint8_t command = USARTD0.DATA;

    /* Get the two MSB of command */
    switch (command >> 6) {

        case 0:
            motor = LEFT_MOTOR;
            direction = BACKWARD;
            break;

        case 1:
            motor = LEFT_MOTOR;
            direction = FORWARD;
            break;

        case 2:
            motor = RIGHT_MOTOR;
            direction = BACKWARD;
            break;

        case 3:
            motor = RIGHT_MOTOR;
            direction = FORWARD;
            break;
    }

    /*
     * Set the left or right motor thrust, with the correct direction and speed.
     *
     * 0x3f = 0b00111111 and masks out the speed from the command.  This
     * speed (0..63) is then scaled to (0..127) by multiplying by two and
     * adding one.  The direction variable is positiv for forward and
     * negative for backward direction, and thus give the final speed as an
     * int8_t of range -127..127
     *
     */
    uart_motor_thrust[motor] = direction*(command & 0x3f)*2 + 1;

    if (uart_receiver_ready == FALSE) {

        /* Skip the first interrupts since the receiver is not stable then */
        if (++number_of_uart_commands >= 100) {
            uart_receiver_ready = TRUE;
        }
    }
}
```

code/ASV_H–bridge_controller/Makefile

```
DEVICE = atxmega32a4
AVRDUDE = avrdude -p x32a4 -c jtag2pdi -P usb
F_CPU=8000000UL

CC = avr-gcc
CFLAGS = -g -W -Wall -O2 -std=gnu99 -mmcu=$(DEVICE) -DF_CPU=$(F_CPU) -lm -pedantic

OBJECTS = main.o

all: main.hex

clean:
	rm -f *.hex *.lst *.o *.bin

$(OBJECTS): | depend

main.bin: $(OBJECTS)
	$(CC) $(CFLAGS) -o main.bin $(OBJECTS)

main.hex: main.bin
	avr-objcopy -j .text -j .data -O ihex main.bin main.hex
	avr-size --totals *.o
	avr-size -C --mcu=$(DEVICE) *.o

read:
	$(AVRDUDE) -U eeprom:r:eeprom.dat:r
	hd eeprom.dat

depend:
	@$(CC) -MM $(ALL_CFLAGS) *.c | sed 's/$$/ Makefile/'

flash: all
	$(AVRDUDE) -U flash:w:main.hex:i
```

Appendix H

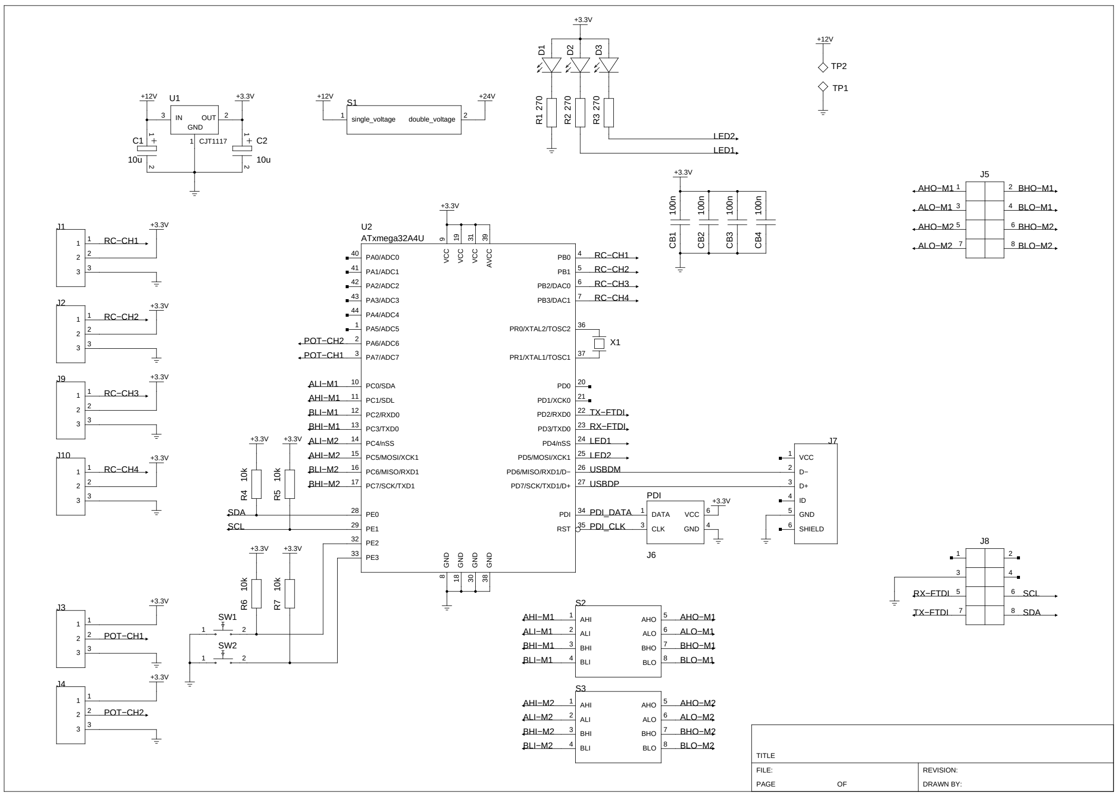
Second H-bridge schematics



Appendix I

H-bridge driver schematics

The schematic for the voltage doubler (subcircuit S1) is shown in figure 1.22. The schematic for the singe H-bridge driver (subcircuit S2 and S3) is shown in figure 1.21.



Appendix J

HydroAHRS mk.I MCU code

Code for ATxmega32A4U in HydroAHRS mk.I, as described in section 2.3.1. The code is licensed under GNU GPL version 2 or later.

code/hydroAHRS_mkI/main.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <inttypes.h>
#include <stdio.h>

#include "main.h"
#include "twi.h"
#include "uart.h"
#include "matrix.h"
#include "MPU9150.h"
#include "imu_calc.h"

void
CCP_write( volatile uint8_t * address, uint8_t value ) {

    /* Begin a critical task, so we must disable interrupt */
    uint8_t volatile saved_sreg = SREG;
    cli();

    volatile uint8_t * tmpAddr = address;
    asm volatile(
    "movw␣r30,␣␣%0"       "\n\t"
    "ldi␣␣r16,␣␣%2"       "\n\t"
    "out␣␣␣%3,␣r16"       "\n\t"
    "st␣␣␣␣␣Z,␣␣%1"       "\n\t"
    :
    : "r" (tmpAddr), "r" (value), "M" (CCP_IOREG_gc), "i" (&CCP)
    : "r16", "r30", "r31"
    );

    /* End the critical task */
    SREG = saved_sreg;
}

void
clock_init(void) {

    /* Select 32 kHz as external clock */
    OSC.XOSCCTRL = OSC_XOSCSEL_32KHz_gc;

    /* Start the 32MHz and external clock, but keep 2MHz clock on */
    OSC.CTRL |= OSC_XOSCEN_bm | OSC_RC32MEN_bm;

    /* Wait until 32MHz clock is ready */
    while( !(OSC.STATUS & OSC_RC32MRDY_bm) );

    /* Wait for the external oscillator to stabilize. */
    while ( !(OSC.STATUS & OSC_XOSCRDY_bm) );

    /* Enable write access to protected register CLK.CTRL with CCP_write
     * Set 32MHz clock as source */
    CCP_write( &CLK.CTRL, CLK_SCLKSEL_RC32M_gc );

    /* Wait for the system to switch clock */
    _delay_us(2);

    /* Disable all clocks except 32MHz and external clock */
    OSC.CTRL = OSC_XOSCEN_bm | OSC_RC32MEN_bm;

    /* Set 32 kHz TOSC as calibration reference */
    OSC.DFLLCTRL = OSC_RC32MCREF_bm;

    /* Enable automatic run-time calibration */
    DFLLRC32M.CTRL = DFLL_ENABLE_bm;
}

void
rtc_init(void) {

   // Set 32 kHz from external 32kHz oscillator as clock source for RTC.
   CLK.RTCCTRL = CLK_RTCSRC_TOSC32_gc | CLK_RTCEN_bm;

   // Wait until RTC is not busy.
   while ( RTC.STATUS & RTC_SYNCBUSY_bm );

   /* Period register value. Must subtract 1, because zero value counted.
    * Gives an overflow every 1/1024 second */
   RTC.PER = 32 - 1;

   /* Make sure COMP and CNT is 0. */
   RTC.COMP = 0;
   RTC.CNT = 0x0000;

   /* Divide by 1, so 32.768 kHz frequency */
   RTC.CTRL = RTC_PRESCALER_DIV1_gc;

   /* Enable overflow interrupt. */
   RTC.INTCTRL = RTC_OVFINTLVL_HI_gc | RTC_COMPINTLVL_OFF_gc;
}

void
int_init() {

    /* Configure PB2 as input */
    PORTB.DIRCLR = PIN2_bm;

    /* Enable INT0 as low level interrupt */
    PORTB.INTCTRL = PORT_INT0LVL_LO_gc;

    /* Assign PB2 as source for INT0 */
    PORTB.INT0MASK = PIN2_bm;

    /* Configure interrupt on falling edge */
    PORTB.PIN2CTRL = PORT_ISC_FALLING_gc;

    /* Enable high and low level interrupt */
    PMIC.CTRL |= PMIC_HILVLEN_bm;
    PMIC.CTRL |= PMIC_LOLVLEN_bm;

    /* Enable global interrupt */
    sei();
}

void
led_init() {

    /* Set Red LED-pin PC2 as output */
    //PORTC.DIRSET = PIN2_bm;

    /* Set Red LED-pin PC2 as input, so it won't light up before we have
     * received the 'start'-signal */
    PORTC.DIRCLR = PIN2_bm;

    /* Set Green LED-pin PC3 as output */
    PORTC.DIRSET = PIN3_bm;

    /* Invert the LED pins, so the LEDs light when we set them high */
    PORTC.PIN2CTRL |= PORT_INVEN_bm;
    PORTC.PIN3CTRL |= PORT_INVEN_bm;
}


int
main() {
    clock_init();
    uart_init();
    rtc_init();
    twi_init();
    int_init();
    led_init();

    /* Clear screen */
    uart_send_str("\x1b[H\x1b[2J");

    _delay_ms(50);

    MPU9150_init();

    for(;;) {
    }
}


ISR(PORTB_INT0_vect) {

    /* Toggle the green LED when we get data from the sensor */
    PORTC.OUTTGL = PIN3_bm;

    /* Turn the red LED on when we begin to compute */
    //PORTC.OUTSET = PIN2_bm;

    read_calculate_send_angles();

    /* Turn the red LED off when we are finished with the compution */
    //PORTC.OUTCLR = PIN2_bm;
}

relative_time_t relative_time;
uint8_t running = FALSE;
uint8_t send_one_measurment = FALSE;

ISR(USARTD0_RXC_vect) {

    uint8_t command;

    command = USARTD0.DATA;

    /* Start sending angles */
    if (command == 's') {
        running = TRUE;
        RTC.CNT = 0x0000;
        TCC0.CNT = 0;
        relative_time.ticks = 0;
        relative_time.secs = 0;
        relative_time.minutes = 0;
        relative_time.hours = 0;
        relative_time.days = 0;
        PORTC.DIRSET = PIN2_bm;
    }

    /* Stop sending angles */
    if (command == 'S') {
        running = FALSE;
        PORTC.DIRCLR = PIN2_bm;
    }

    /* Set gyro bias */
    if (command == 'g') {
        set_gyro_bias();
    }

    /* Set angle reference */
    if (command == 'a') {
        set_angle_reference();
    }

    /* Send one measurment */
    if (command == '1') {
        RTC.CNT = 0x0000;
        TCC0.CNT = 0;
        relative_time.ticks = 0;
        relative_time.secs = 0;
        relative_time.minutes = 0;
        relative_time.hours = 0;
        relative_time.days = 0;
        send_one_measurment = TRUE;
    }
}

ISR(RTC_OVF_vect) {

    /* Ticks is 1/1024 seconds */
    relative_time.ticks++;

    if (relative_time.ticks > 1023) {
        relative_time.ticks = 0;
        relative_time.secs++;

        PORTC.OUTTGL = PIN2_bm;
    }

    if (relative_time.secs > 59) {
        relative_time.secs = 0;
        relative_time.minutes++;
    }

    if (relative_time.minutes > 59) {
        relative_time.minutes = 0;
        relative_time.hours++;
    }

    if (relative_time.hours > 23) {
        relative_time.hours = 0;
        relative_time.days++;
    }
}
```

code/hydroAHRS_mkI/main.h

```
# ifndef MAIN_H
# define MAIN_H

#define TRUE 1
#define FALSE 0

typedef struct {
    uint16_t ticks;
    uint8_t secs;
    uint8_t minutes;
    uint8_t hours;
    uint16_t days;

```

```c
} relative_time_t;

extern relative_time_t relative_time;
extern uint8_t running;
extern uint8_t send_one_measurment;

#endif
```

code/hydroAHRS_mkI/imu_calc.c

```c
#include <math.h>
#include <stdlib.h>
#include <inttypes.h>

#include "main.h"
#include "imu_calc.h"
#include "matrix.h"
#include "twi.h"
#include "uart.h"
#include "MPU9150.h"

static void get_sensor_data(raw_sensor_t *raw_sensor_data);
static void remove_gyro_bias(int8_t *gyro_bias, raw_sensor_t *raw_sensor_data);
static void convert_sensor_data(raw_sensor_t *raw_sensor_data, sensor_t *sensor_data);
static void apply_drift_correction(float *omega_correction_P, float *omega_correction_I
    , sensor_t *sensor_data);
static void update_rotation_matrix(sensor_t *sensor_data, float R[][3]);
static void normalize_rotation_matrix(float R[][3]);
static void calculate_roll_pitch_error(sensor_t *sensor_data, float R[][3], float *
    roll_pitch_error);
static void calculate_omega_correction_P(float *roll_pitch_error, float *
    omega_correction_P);
static void calculate_omega_correction_I(float *roll_pitch_error, float *
    omega_correction_I);
static void retrieve_euler_angles(axes_t *angle, float R[][3]);
static void send_angles(raw_sensor_t *raw_sensor_data, sensor_t *sensor_data, axes_t *
    angle, relative_time_t *relative_time);
static void send_raw_data(raw_sensor_t *raw_sensor_data);
static int16_t comp2int(uint8_t msb, uint8_t lsb);

raw_sensor_t raw_sensor_data;
sensor_t sensor_data;
axes_t angle;

static float R[3][3] = {{1, 0, 0},
                        {0, 1, 0},
                        {0, 0, 1}};

static int8_t gyro_bias[3] = {12, 24, 5};
static float angle_reference[3] = {0.0, 0.0, 0.0};

static float omega_correction_P[3];
static float omega_correction_I[3];

float roll_pitch_error[3];

extern void
read_calculate_send_angles() {


    get_sensor_data(&raw_sensor_data);

    // send_raw_data(&raw_sensor_data);

    remove_gyro_bias(gyro_bias, &raw_sensor_data);

    convert_sensor_data(&raw_sensor_data, &sensor_data);

    apply_drift_correction(omega_correction_P, omega_correction_I, &sensor_data);

    update_rotation_matrix(&sensor_data, R);

    normalize_rotation_matrix(R);

    /* Calculate drift correction */
    calculate_roll_pitch_error(&sensor_data, R, roll_pitch_error);

    calculate_omega_correction_P(roll_pitch_error, omega_correction_P);

    calculate_omega_correction_I(roll_pitch_error, omega_correction_I);

    retrieve_euler_angles(&angle, R);

    if (send_one_measurment == TRUE) {
        send_angles(&raw_sensor_data, &sensor_data, &angle, &relative_time);
        //uart_send_str("\x04");
        send_one_measurment = FALSE;

    } else if (running == TRUE) {
        send_angles(&raw_sensor_data, &sensor_data, &angle, &relative_time);
    }
}

/*
 * Get the data from the accelerometer and gyroscope
 * ------
 *
 * Output: Accelerometer raw data
 * Output: Gyroscope raw data
 */
static void
get_sensor_data(raw_sensor_t *raw_sensor_data) {

    i2c_slave_addr = MPU9150_ADDR;

    raw_sensor_data->temperature = (read_reg(TEMP_OUT_H)<<8) + read_reg(TEMP_OUT_L);

    raw_sensor_data->accl_x = comp2int(read_reg(ACCEL_XOUT_H), read_reg(ACCEL_XOUT_L));
    raw_sensor_data->accl_y = comp2int(read_reg(ACCEL_YOUT_H), read_reg(ACCEL_YOUT_L));
    raw_sensor_data->accl_z = comp2int(read_reg(ACCEL_ZOUT_H), read_reg(ACCEL_ZOUT_L));

    raw_sensor_data->gyro_x = comp2int(read_reg(GYRO_XOUT_H), read_reg(GYRO_XOUT_L));
    raw_sensor_data->gyro_y = comp2int(read_reg(GYRO_YOUT_H), read_reg(GYRO_YOUT_L));
    raw_sensor_data->gyro_z = comp2int(read_reg(GYRO_ZOUT_H), read_reg(GYRO_ZOUT_L));

    i2c_slave_addr = AK8975C_ADDR;
    write_reg(AK_CNTL, 0x01);

    raw_sensor_data->magn_x = comp2int(read_reg(AK_HXH), read_reg(AK_HXL));
    raw_sensor_data->magn_y = comp2int(read_reg(AK_HYH), read_reg(AK_HYL));
    raw_sensor_data->magn_z = comp2int(read_reg(AK_HZH), read_reg(AK_HZL));
}

inline void
set_gyro_bias() {

    // TODO: Average over multiple measurments
    gyro_bias[0] = raw_sensor_data.gyro_x;
    gyro_bias[1] = raw_sensor_data.gyro_y;
    gyro_bias[2] = raw_sensor_data.gyro_z;

    char str_buf[30];

    uart_send_str("gyro_x␣bias:␣");
    itoa(gyro_bias[0], str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str(",␣gyro_y␣bias:␣");
    itoa(gyro_bias[1], str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str(",␣gyro_z␣bias:␣");
    itoa(gyro_bias[2], str_buf, 10);
    uart_send_str(str_buf);

```

```c
    uart_send_str("\n\r");
}

/*
 * Remove the raw bias from the gyroscope data
 * -----
 *
 * Input: Gyroscope raw data with bias
 * Output: Gyroscope raw data without bias
 */
static inline void
remove_gyro_bias(int8_t *gyro_bias, raw_sensor_t *raw_sensor_data) {

    raw_sensor_data->gyro_x = raw_sensor_data->gyro_x - gyro_bias[0];
    raw_sensor_data->gyro_y = raw_sensor_data->gyro_y - gyro_bias[1];
    raw_sensor_data->gyro_z = raw_sensor_data->gyro_z - gyro_bias[2];
}

static void
convert_sensor_data(raw_sensor_t *raw_sensor_data, sensor_t *sensor_data) {

    /* Convert accelerometer and gyroscope data to real units */

    /* Get acceleration in mg */
    sensor_data->accl_x = (float) raw_sensor_data->accl_x / ACC_LSB;
    sensor_data->accl_y = (float) raw_sensor_data->accl_y / ACC_LSB;
    sensor_data->accl_z = (float) raw_sensor_data->accl_z / ACC_LSB;

    /* Get angular rotaion in deg/sec */
    /* TODO: Make sure raw_sensor_data should be divided on GYRO_HZ */
    sensor_data->gyro_x = (float) raw_sensor_data->gyro_x / (GYRO_LSB * GYRO_HZ);
    sensor_data->gyro_y = (float) raw_sensor_data->gyro_y / (GYRO_LSB * GYRO_HZ);
    sensor_data->gyro_z = (float) raw_sensor_data->gyro_z / (GYRO_LSB * GYRO_HZ);

    /* Convert gyroscope data from degrees to radians */
    sensor_data->gyro_x = sensor_data->gyro_x * DEG2RAD;
    sensor_data->gyro_y = sensor_data->gyro_y * DEG2RAD;
    sensor_data->gyro_z = sensor_data->gyro_z * DEG2RAD;

    /* Keep gyro_z at 0 untill the magnetometer is implemented */
    sensor_data->gyro_z = 0;

    /* Get temperatur in degrees Celcius */
    sensor_data->temperature = (raw_sensor_data->temperature + 11900) / 340;
}


/*
 * Add the drift correction to the angular change from last measurmemt
 * -----
 *
 * Input: Measured change in angle (in degrees) since last measurment
 * Input: Omegea_p
 * Input: Omegea_i
 * Output: Corrected measured change in angle since last measurment
 */
static inline void
apply_drift_correction(float *omega_correction_P, float *omega_correction_I, sensor_t *
     sensor_data) {

    sensor_data->gyro_x = sensor_data->gyro_x + omega_correction_P[0] +
         omega_correction_I[0];
    sensor_data->gyro_y = sensor_data->gyro_y + omega_correction_P[1] +
         omega_correction_I[1];
    sensor_data->gyro_z = sensor_data->gyro_z + omega_correction_P[2] +
         omega_correction_I[2];
}

/*
 * Make a matrix consisting of the accelerometer corrected change in degrees
 * since last measurment (that we got from the accelerometer)
 *
 * Update the rotation matrix by multiplying it with the update matrix
 * "Add the change in degrees from last measurment to the current state"
 *
 */
static void
update_rotation_matrix(sensor_t *sensor_data, float R[][3]) {

    float R_buffer[3][3];

    /* Construct update matrix */
    float R_update[3][3] = {{                    1, -sensor_data->gyro_z,  sensor_data
        ->gyro_y },
                            {  sensor_data->gyro_z,                    1, -sensor_data
                                ->gyro_x },
                            { -sensor_data->gyro_y,  sensor_data->gyro_x,
                                                   1 }};

    /* Multiply "rotation matrix" with "update matrix" */

    /* R_buffer = R * Rup */
    matrix_mult(R_buffer, R, R_update);

    /* R = R_buffer */
    matrix_copy(R, R_buffer);
}

static float
root_sum_square(float x, float y, float z) {
    return sqrt( square(x) + square(y) + square(z) );
}

static void
normalize_vector(float *vector) {

    float rss = root_sum_square(vector[0], vector[1], vector[2]);

    vector[0] = vector[0] / rss;
    vector[1] = vector[1] / rss;
    vector[2] = vector[2] / rss;
}

/*
 * Renormalize the rotation matrix
 *
 * In a perfect world then the rotation matrix would remain ortonormal,
 * but since it is not, we have to correct for this.
 * ------
 *
 * Input: Rotation matrix
 * Output: Orthonormal rotation matrix
 */
static void
normalize_rotation_matrix(float R[][3]) {

    float R_buffer[3][3];

    /* Renormalize rotation matrix (to keep orthogonal) */

    /* When the dot product of two vectors is zero, then they are normal to
     * each other. If not, then they are not normal. We use this to form an
     * error estimate for non-normalized vectors.
     */

    /* Dot product */
    /* TODO: Make this a function */
    float error = -0.5 * (R[0][0] * R[1][0] +
                          R[0][1] * R[1][1] +
                          R[0][2] * R[1][2]);

    /* We apply the error scalar to the two rows in opposite direction by cross
     * coupling */
    R_buffer[0][0] = (R[0][0] * error) + R[0][0];
    R_buffer[0][1] = (R[0][1] * error) + R[0][1];
    R_buffer[0][2] = (R[0][2] * error) + R[0][2];
    R_buffer[1][0] = (R[1][0] * error) + R[1][0];
    R_buffer[1][1] = (R[1][1] * error) + R[1][1];
    R_buffer[1][2] = (R[1][2] * error) + R[1][2];

    /* Cross product */
    R_buffer[2][0] = ( (R_buffer[0][1] * R_buffer[1][2]) - (R_buffer[0][2] * R_buffer
         [1][1]) );
    R_buffer[2][1] = ( (R_buffer[0][2] * R_buffer[1][0]) - (R_buffer[0][0] * R_buffer
         [1][2]) );
    R_buffer[2][2] = ( (R_buffer[0][0] * R_buffer[1][1]) - (R_buffer[0][1] * R_buffer
         [1][0]) );

    normalize_vector(R_buffer[0]);
    normalize_vector(R_buffer[1]);
    normalize_vector(R_buffer[2]);

    /* R = R_buffer */
    matrix_copy(R, R_buffer);

}

/* Error estimate
 * -----
 *
 * Input = normalized rotation matrix
 * Input = normalized acceleration vector
 * Output = Error estimate for roll and pitch (rpe)
 */
static void
calculate_roll_pitch_error(sensor_t *sensor_data, float R[][3], float *roll_pitch_error
    ) {

    float g_magnitude;
    float g_vector[3];

    g_magnitude = root_sum_square(sensor_data->accl_x, sensor_data->accl_y, sensor_data
         ->accl_z);

    g_vector[0] = sensor_data->accl_x / g_magnitude;
    g_vector[1] = sensor_data->accl_y / g_magnitude;
    g_vector[2] = sensor_data->accl_z / g_magnitude;

    /* Cross product */
    /* TODO: cross product function */
    roll_pitch_error[0] = ((g_vector[1] * R[2][2]) - (g_vector[2] * R[2][1])) / 65536L;
    roll_pitch_error[1] = ((g_vector[2] * R[2][0]) - (g_vector[0] * R[2][2])) / 65536L;
    roll_pitch_error[2] = ((g_vector[0] * R[2][1]) - (g_vector[1] * R[2][0])) / 65536L;

}

static inline void
calculate_omega_correction_P(float *roll_pitch_error, float *omega_correction_P) {

    /* Calculate omega correction P */
    omega_correction_P[0] = roll_pitch_error[0] * K_P_PITCH_ROLL;
    omega_correction_P[1] = roll_pitch_error[1] * K_P_PITCH_ROLL;
    omega_correction_P[2] = roll_pitch_error[2] * K_P_PITCH_ROLL;
}

static inline void
calculate_omega_correction_I(float *roll_pitch_error, float *omega_correction_I) {

    static float acceleration_integral[3] = {0,0,0};

    /* Calculate omega correction I */

    /* */
    acceleration_integral[0] = acceleration_integral[0] + (roll_pitch_error[0] *
        K_I_PITCH_ROLL) / (1024L * GYRO_HZ);
    acceleration_integral[1] = acceleration_integral[1] + (roll_pitch_error[1] *
        K_I_PITCH_ROLL) / (1024L * GYRO_HZ);
    acceleration_integral[2] = acceleration_integral[2] + (roll_pitch_error[2] *
        K_I_PITCH_ROLL) / (1024L * GYRO_HZ);

    omega_correction_I[0] = acceleration_integral[0];
    omega_correction_I[1] = acceleration_integral[1];
    omega_correction_I[2] = acceleration_integral[2];
}

static inline void
retrieve_euler_angles(axes_t *angle, float R[][3]) {

    angle->roll  = ( atan2(-R[2][0], R[2][2]) ) * RAD2DEG;
    angle->pitch = ( asin(R[2][1])            ) * RAD2DEG;
    angle->yaw   = ( atan2(-R[0][1], R[1][1]) ) * RAD2DEG;
}

static void
send_angles(raw_sensor_t *raw_sensor_data, sensor_t *sensor_data, axes_t *angle,
     relative_time_t *relative_time) {

    float scaled_roll;
    float scaled_pitch;
    char str_buf[30];

    sprintf(str_buf, "%0.2d", relative_time->hours);
    uart_send_str(str_buf);
    sprintf(str_buf, ":%0.2d", relative_time->minutes);
    uart_send_str(str_buf);
    sprintf(str_buf, ":%0.2d", relative_time->secs);
    uart_send_str(str_buf);
    sprintf(str_buf, ".%0.3d", (relative_time->ticks*1000UL)/1024UL);
    uart_send_str(str_buf);
    uart_send_str(",");

    sprintf(str_buf, "%+0.5d,", raw_sensor_data->accl_x);
    uart_send_str(str_buf);

    sprintf(str_buf, "%+0.5d,", raw_sensor_data->accl_y);
    uart_send_str(str_buf);

    sprintf(str_buf, "%+0.5d,", raw_sensor_data->accl_z);
    uart_send_str(str_buf);

    sprintf(str_buf, "%+0.5d,", raw_sensor_data->gyro_x);
    uart_send_str(str_buf);

    sprintf(str_buf, "%+0.5d,", raw_sensor_data->gyro_y);
    uart_send_str(str_buf);

    sprintf(str_buf, "%+0.5d,", raw_sensor_data->gyro_z);
    uart_send_str(str_buf);

    sprintf(str_buf, "%+0.2d,", sensor_data->temperature);
    uart_send_str(str_buf);

    scaled_roll = angle->roll/1.1 - angle_reference[0]/1.1;

    dtostrf(scaled_roll, 3, 2, str_buf);
    uart_send_str(str_buf);
    uart_send_str(",");

    scaled_pitch = angle->pitch/1.1 - angle_reference[1]/1.1;

    dtostrf(scaled_pitch, 3, 2, str_buf);
    uart_send_str(str_buf);

    uart_send_str("\n\r");
}

inline void
set_angle_reference() {

    angle_reference[0] = angle.roll;
    angle_reference[1] = angle.pitch;
    angle_reference[2] = angle.yaw;
```

```c

    char str_buf[30];

    uart_send_str("roll reference: ");
    dtostrf(angle_reference[0], 3, 2, str_buf);
    uart_send_str(str_buf);

    uart_send_str(", pitch reference: ");
    dtostrf(angle_reference[1], 3, 2, str_buf);
    uart_send_str(str_buf);

    uart_send_str("\n\r");
}


static void
send_raw_data(raw_sensor_t *raw_sensor_data) {

    char str_buf[30];

    uart_send_str("Temp: ");
    itoa(raw_sensor_data->temperature, str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str("\tAccl_x: ");
    itoa(raw_sensor_data->accl_x, str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str("\tAccl_y: ");
    itoa(raw_sensor_data->accl_y, str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str("\tAccl_z: ");
    itoa(raw_sensor_data->accl_z, str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str("\tGyro_x: ");
    itoa(raw_sensor_data->gyro_x, str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str("\tGyro_y: ");
    itoa(raw_sensor_data->gyro_y, str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str("\tGyro_z: ");
    itoa(raw_sensor_data->gyro_z, str_buf, 10);
    uart_send_str(str_buf);

    uart_send_str("\n\r");
}

static int16_t
comp2int(uint8_t msb, uint8_t lsb) {

    int16_t x = 0;

    if ((msb & 0x80) == 0x80) {

        lsb ^= 0xff;
        msb ^= 0xff;
        msb &= 0x7f;

        x = (int16_t) ((msb << 8) + lsb + 1);

        return -x;

    } else {

        x = (int16_t) ((msb << 8) + lsb);

        return x;
    }
}
```

code/hydroAHRS_mkI/imu_calc.h

```c
#ifndef IMU_CALC_H
#define IMU_CALC_H

#define DEG2RAD (M_PI/180L)
#define RAD2DEG (180L/M_PI)

#define K_I_PITCH_ROLL 512
#define K_P_PITCH_ROLL 512

typedef struct {
    int16_t accl_x;
    int16_t accl_y;
    int16_t accl_z;

    int16_t gyro_x;
    int16_t gyro_y;
    int16_t gyro_z;

    int16_t magn_x;
    int16_t magn_y;
    int16_t magn_z;

    int16_t temperature;
} raw_sensor_t;

typedef struct {
    float accl_x;
    float accl_y;
    float accl_z;

    float gyro_x;
    float gyro_y;
    float gyro_z;

    float magn_x;
    float magn_y;
    float magn_z;

    int16_t temperature;
} sensor_t;

typedef struct {
    float roll;
    float pitch;
    float yaw;
} axes_t;

extern void read_calculate_send_angles();
extern void set_gyro_bias();
extern void set_angle_reference();

#endif
```

code/hydroAHRS_mkI/uart.c

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

#include "uart.h"

void
uart_init(void) {

    /* Set PD3 (TX) as output */
    PORTD.DIRSET = PIN3_bm;

    /* Set the baud rate */
    // USARTC1.BAUDCTRLA = (uint8_t) BAUD;
    // USARTC1.BAUDCTRLB = ((USARTC1.BAUDCTRLB) & 0xF0) | ((BAUD >> 8) & 0x0F);
    USARTD0.BAUDCTRLA = 0x2E;
    USARTD0.BAUDCTRLB = 0x98;

    /* Enable low level UART receive interrupt */
    USARTD0.CTRLA = USART_RXCINTLVL_LO_gc;

    /* Enable the UART transmitter and receiver */
    USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

    /* Asynchronous USART, no parity, 1 stop bit, 8 data bits */
    USARTD0.CTRLC = USART_CHSIZE0_bm | USART_CHSIZE1_bm;
}

void
uart_send_byte(uint8_t ch) {

    /* Wait for TX buffer to be empty,
     * which is indicated by the UDRE1 bit being cleared */
    while (!(USARTD0.STATUS & USART_DREIF_bm)) {
    }

    USARTD0.DATA = ch;
}

void
uart_send_hex(uint8_t ch) {
    uint8_t temp;

    uart_send_str("0x");

    /* Send high nibble */
    temp = (uint8_t) ((ch & 0xF0) >> 4 ) + 0x30;

    if (temp > 0x39)
        temp += 7;

    uart_send_byte(temp);

    /* Send low nibble */
    temp = (ch & 0x0F) + 0x30;

    if (temp > 0x39)
        temp += 7;

    uart_send_byte(temp);
}

void
uart_send_str(char *ptr) {

    /* Send the string, one byte at time */
    while(*ptr) {
        uart_send_byte(*ptr);
        ptr++;
    }
}

void
uart_send_newline(void) {

    /* Send newline and return */
    uart_send_byte('\n');
    uart_send_byte('\r');
}
```

code/hydroAHRS_mkI/uart.h

```
#ifndef UART_H
#define UART_H

#define BAUD ((uint16_t) ((F_CPU / (16.0 * (115200))) + 0.5) - 1)

void uart_init(void);
void uart_send_byte(uint8_t ch);
void uart_send_hex(uint8_t ch);
void uart_send_str(char *ptr);
void uart_send_newline(void);

#endif
```

code/hydroAHRS_mkI/twi.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

#include "main.h"
#include "uart.h"
#include "twi.h"

volatile int8_t TWIBusy;    // This flag is set when a send or receive is started
volatile int8_t byte2Send; // Number of bytes to send to DS1337
volatile int8_t byte2Read; // Number of bytes to read from DS1337
volatile int8_t status;     // Transaction status

volatile uint8_t wIndex;    // index to txBuf
volatile uint8_t rIndex;    // index to rxBuf

int8_t rxBuf[15];           // Buffer to hold time read from keyboard
int8_t txBuf[10];           // Buffer of time for transfer to RTC
int8_t dispBuf[22];         // Display buffer

uint8_t i2c_slave_addr; // = MPU9150_ADDR;

void
twi_init() {
    /* Configure SDA as output */
    PORTC.DIRSET = PIN0_bm;

    /* Configure SCL as output */
    PORTC.DIRSET = PIN1_bm;

    /* Enable TWI Master and (high level) TWI read and write interrupt */
    TWIC.MASTER.CTRLA = TWI_MASTER_ENABLE_bm | TWI_MASTER_RIEN_bm | TWI_MASTER_WIEN_bm;
    TWIC.MASTER.CTRLA |= TWI_MASTER_INTLVL0_bm | TWI_MASTER_INTLVL1_bm;

    /* Set the TWI frequency */
    TWIC_MASTER_BAUD = TWI_FREQ;

    /* Force TWI bus to idle state */
    TWIC.MASTER.STATUS = TWI_MASTER_BUSSTATE0_bm;
}

uint8_t
read_reg(uint8_t addr) {

    /* Wait until TWI bus is idle */
    while (TWIBusy);

    /* Occupy the TWI bus */
    TWIBusy = 1;

    status = NORMAL;
    wIndex = 0;
    rIndex = 0;

    byte2Send = 1;
    byte2Read = 1;

    txBuf[0] = addr;

    /* Generate START condition and send SLA + W */
    TWIC.MASTER.ADDR = i2c_slave_addr + 0;

    /* Wait until read is complete */
    while (TWIBusy);

    return rxBuf[0];
}

void
write_reg(uint8_t addr, uint8_t value) {

    /* Wait until TWI bus is idle */
    while (TWIBusy);

    /* Occupy the TWI bus */
    TWIBusy = 1;

    status = NORMAL;
    wIndex = 0;
    rIndex = 0;

    byte2Send = 2;
    byte2Read = 0;

    txBuf[0] = addr;
    txBuf[1] = value;

    /* Generate START condition and send SLA + W */
    TWIC.MASTER.ADDR = i2c_slave_addr + 0;

    /* Wait until read is complete */
    while (TWIBusy);
}

ISR(TWIC_TWIM_vect) {

    /* Arbitration lost condition */
    if (TWIC.MASTER.STATUS & TWI_MASTER_ARBLOST_bm) {
        status = ARBLOST;

    /* Bus error condition */
    } else if (TWIC.MASTER.STATUS & TWI_MASTER_BUSERR_bm) {
        status = BUSERR;

    /* Send data */
    } else if (TWIC.MASTER.STATUS & TWI_MASTER_WIF_bm) {

        if (TWIC.MASTER.STATUS & TWI_MASTER_RXACK_bm) {

            /* Receive negative ACK from slave */
            status = NEGACK;

            /* Generate a STOP condition */
            TWIC.MASTER.CTRLC = 0x03;

            /* Release TWI bus */
            TWIBusy = 0;

        /* More data to send? */
        } else if (byte2Send > 0) {

            /* Send out next byte */
            TWIC.MASTER.DATA = txBuf[wIndex++];
            byte2Send--;

        /* All data bytes have been transmitted */
        } else {

            /* Need to read data? */
            if (byte2Read > 0) {

                /* generate RESTART & send SLA + R */
                //TWIC.MASTER.ADDR = MPU9150_ADDR + 1;
                TWIC.MASTER.ADDR = i2c_slave_addr + 1;

                /* All transaction is finished */
            } else {

                /* generate STOP condition */
                TWIC.MASTER.CTRLC = 0x03;

                /* normal result */
                status = NORMAL;

                /* release TWI bus */
                TWIBusy = 0;
            }
        }

    /* Read data */
    } else if (TWIC.MASTER.STATUS & TWI_MASTER_RIF_bm) {

        if (byte2Read > 1) {

            /* Send ACK and receive the next byte */
            TWIC.MASTER.CTRLC = 0x02;

        } else {

            /* Send NACK and generate STOP condition */
            TWIC.MASTER.CTRLC = 0x07;

            /* Release TWI bus */
            TWIBusy = 0;
        }

        rxBuf[rIndex++] = TWIC.MASTER.DATA;
        byte2Read--;

    } else {

        /* Send NACK and generate STOP condition */
        TWIC.MASTER.CTRLC = 0x07;

        /* Release TWI bus */
        TWIBusy = 0;
    }
}
```

code/hydroAHRS_mkI/twi.h

```
# ifndef TWI_H
# define TWI_H

#define TWI_FREQ ((F_CPU / (2.0 * (400000))) - 5)

#define NORMAL   0  // Normal successful data transfer
#define ARBLOST  1  // Arbitration lost condition
#define BUSERR   2  // Bus error condition
#define NEGACK   3  // Receive NACK condition

void twi_init(void);
//void send_cmd(uint8_t addr, uint8_t value);
uint8_t read_reg(uint8_t addr);
void write_reg(uint8_t addr, uint8_t value);

extern int8_t rxBuf[];
extern uint8_t i2c_slave_addr;

#endif
```

code/hydroAHRS_mkI/matrix.c

```
#include <inttypes.h>

#include "matrix.h"

void
```

```c
matrix_mult(float x[][3], float y[][3], float z[][3]) {

    for (uint8_t i = 0; i < 3; i++) {

        for(uint8_t j = 0; j < 3; j++) {

            x[i][j] = 0;

            for(uint8_t k = 0; k < 3; k++)

                x[i][j] += y[i][k] * z[k][j];
        }
    }
}

void
matrix_copy(float x[][3], float y[][3]) {

    for (uint8_t i = 0; i < 3; i++) {

        for(uint8_t j = 0; j < 3; j++) {

            x[i][j] = y[i][j];
        }
    }
}
```

code/hydroAHRS_mkI/matrix.h

```c
#ifndef MATRIX_H
#define MATRIX_H

void matrix_mult(float x[][3], float y[][3], float z[][3]);
void matrix_copy(float x[][3], float y[][3]);

#endif
```

code/hydroAHRS_mkI/MPU9150.c

```c
#include <inttypes.h>

#include "twi.h"
#include "MPU9150.h"

void
MPU9150_init() {

    i2c_slave_addr = MPU9150_ADDR;

    /* Wake up */
    write_reg(PWR_MGMT_1, 0x00);

    /* Set samplerate to 50 Hz */
    write_reg(SMPLRT_DIV, SAMPLERATEDIVIDER);

    /* Set low pass filter */
    write_reg(CONFIG, DLPF_CFG);

    /* Interrupt when new sample is ready */
    write_reg(INT_ENABLE, 0x01);

    /* I2C Pass-by */
    write_reg(INT_PIN_CFG, BYPASS_CFG);
}
```

code/hydroAHRS_mkI/MPU9150.h

```c
#ifndef MPU9150_H
#define MPU9150_H

```

```
#define MPU9150_ADDR     0xD0
#define AK8975C_ADDR     0x18

/* Definitions of register MPU9150 map */
#define AUX_VDDIO              1
#define SMPLRT_DIV            25
#define CONFIG                26
#define GYRO_CONFIG           27
#define ACCEL_CONFIG          28
#define FF_THR                29
#define FF_DUR                30
#define MOT_THR               31
#define MOT_DUR               32
#define ZRMOT_THR             33
#define ZRMOT_DUR             34
#define FIFO_EN               35
#define I2C_MST_CTRL          36
#define I2C_SLV0_ADDR         37
#define I2C_SLV0_REG          38
#define I2C_SLV0_CTRL         39
#define I2C_SLV1_ADDR         40
#define I2C_SLV1_REG          41
#define I2C_SLV1_CTRL         42
#define I2C_SLV2_ADDR         43
#define I2C_SLV2_REG          44
#define I2C_SLV2_CTRL         45
#define I2C_SLV3_ADDR         46
#define I2C_SLV3_REG          47
#define I2C_SLV3_CTRL         48
#define I2C_SLV4_ADDR         49
#define I2C_SLV4_REG          50
#define I2C_SLV4_DO           51
#define I2C_SLV4_CTRL         52
#define I2C_SLV4_DI           53
#define I2C_MST_STATUS        54
#define INT_PIN_CFG           55
#define INT_ENABLE            56
#define INT_STATUS            58
#define ACCEL_XOUT_H          59
#define ACCEL_XOUT_L          60
#define ACCEL_YOUT_H          61
#define ACCEL_YOUT_L          62
#define ACCEL_ZOUT_H          63
#define ACCEL_ZOUT_L          64
#define TEMP_OUT_H            65
#define TEMP_OUT_L            66
#define GYRO_XOUT_H           67
#define GYRO_XOUT_L           68
#define GYRO_YOUT_H           69
#define GYRO_YOUT_L           70
#define GYRO_ZOUT_H           71
#define GYRO_ZOUT_L           72

/* External data registers */
#define MOT_DETECT_STAT_US    97
#define I2C_SLV0_DO           99
#define I2C_SLV1_DO          100
#define I2C_SLV2_DO          101
#define I2C_SLV3_DO          102
#define I2C_MST_DELAY_CT_RL  103
#define SIGNAL_PATH_RES_ET   104
#define MOT_DETECT_CTRL      105
#define USER_CTRL            106
#define PWR_MGMT_1           107
#define PWR_MGMT_2           108
#define FIFO_COUNTH          114
#define FIFO_COUNTL          115
#define FIFO_R_W             116
#define WHO_AM_I             117

/* Register map for AK8975C Magnetometer */
#define AK_INFO   1
#define AK_ST1    2
```

```
#define AK_HXL    3
#define AK_HXH    4
#define AK_HYL    5
#define AK_HYH    6
#define AK_HZL    7
#define AK_HZH    8
#define AK_ST2    9
#define AK_CNTL 10
#define AK_ASAX 16
#define AK_ASAY 17
#define AK_ASAZ 18  // Z-Axis sensitivity

/* MPU9150 Device options */
#define SAMPLERATEDIVIDER 39    // 1kHz/40 = 25Hz sample rate
#define DLPF_CFG           1    // Gives 1kHz sampling and 188Hz BW
#define BYPASS_CFG         2    // Set up for i2c by-pass mode

/* AK8975C Device options */
#define AK_ADR 12              // Slave adress of magnetometer
#define AK_CNTL_MODE 1         // Single measurement mode

/* General options */
#define DEBUG_MODE 1           // If 1, program will be verbose
#define ACC_LSB 16384L          // No. of bits pr g
#define GYRO_LSB 131L           // No. of bits pr deg/s
#define GYRO_HZ 25             // Data output rate
//#define GYRO_DT ((1/GYRO_HZ)*1000)
#define GYRO_DT 40

void MPU9150_init(void);

#endif
```

code/hydroAHRS_mkI/Makefile

```
DEVICE = atxmega32a4
AVRDUDE = avrdude -p x32a4 -c jtag2pdi -P usb
F_CPU=32000000UL

CC = avr-gcc
CFLAGS = -g -W -Wall -O2 -std=gnu99 -mmcu=$(DEVICE) -DF_CPU=$(F_CPU) -lm -pedantic

OBJECTS = main.o uart.o twi.o matrix.o MPU9150.o imu_calc.o

flash:  all
	$(AVRDUDE) -U flash:w:main.hex:i

all:    main.hex

clean:
	rm -f *.hex *.lst *.o *.bin

$(OBJECTS): | depend

main.bin: $(OBJECTS)
	$(CC) $(CFLAGS) -o main.bin $(OBJECTS)

main.hex: main.bin
	avr-objcopy -j .text -j .data -O ihex main.bin main.hex
	avr-size --totals *.o
	#avr-size -C --mcu=$(DEVICE) *.o

read:
	$(AVRDUDE) -U eeprom:r:eeprom.dat:r
	hd eeprom.dat

depend:
	@$(CC) -MM $(ALL_CFLAGS) *.c | sed 's/$$/ Makefile/'
```

Appendix K

HydroAHRS mk.II MCU code

Code for ATxmega32A4U in HydroAHRS mk.II, as described in section 2.3.2.

This code depends on LUFA-130901 from Dean Camera, eMPL from InvenSense and Two Wire Interface (TWI)-drivers from Atmel. LUFA-130901 is available at `http://www.github.com/abcminiuser/lufa/archive/LUFA-130901.zip`. eMPL is available at `http://www.invensense.com/developers/`. TWI drivers is available at `http://www.atmel.com/tools/AVRSOFTWAREFRAMEWORK.aspx`. The code is licensed under the MIT license (listing D.1) unless otherwise specified.

code/hydroAHRS_mkII/main.c

```c
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>
#include <util/delay.h>
#include <inttypes.h>
#include <stdio.h>

#include "main.h"
#include "twi.h"
#include "uart.h"
#include "mpu9150.h"
#include "usb_talk.h"
#include "nmea.h"

#define LED_PORT PORTC
#define LED_PIN PIN2_bm

relative_time_t relative_time;
uint8_t running = FALSE;
uint8_t send_one_measurment = FALSE;
uint8_t getting_angle_reference = FALSE;

//static float avarage_pan = 0.0;

static int calibrate_mag = 0;
static int calibrate_accel = 0;
//static int change = 0;

typedef struct {
    uint8_t calibrated;
    int16_t minVal[3];
    int16_t maxVal[3];
} accel_cal_st;

typedef struct {
    uint8_t calibrated;
    int16_t minVal[3];
    int16_t maxVal[3];
} mag_cal_st;
```

```

/* Create variable in EEPROM */
accel_cal_st EEMEM EEaccel_cal;
mag_cal_st EEMEM EEmag_cal;

char EEMEM EEserial_number[5];

/* Create variable in RAM */
accel_cal_st accel_cal;
mag_cal_st mag_cal;

char serial_number[5] = "0011";

caldata_t accel_cal_or;
caldata_t mag_cal_or;

void(* start_bootloader)(void) = (void (*)(void))(BOOT_SECTION_START/2+0x1FC/2);

int get_ms(unsigned long *count) {
    return 0;
}

void __no_operation(void) { }

void
CCP_write( volatile uint8_t * address, uint8_t value ) {

    /* Begin a critical task, so we must disable interrupt */
    uint8_t volatile saved_sreg = SREG;
    cli();

    volatile uint8_t * tmpAddr = address;
    asm volatile(
    "movw r30,  %0"       "\n\t"
    "ldi  r16,  %2"       "\n\t"
    "out   %3, r16"       "\n\t"
    "st     Z,  %1"       "\n\t"
    :
    : "r" (tmpAddr), "r" (value), "M" (CCP_IOREG_gc), "i" (&CCP)
    : "r16", "r30", "r31"
    );

    /* End the critical task */
    SREG = saved_sreg;
}

void clock_init() {

    /* Start the PLL to multiply the 2MHz RC oscillator to 32MHz */
    OSC.PLLCTRL = OSC_PLLSRC_RC2M_gc | (F_CPU / 2000000);
    OSC.CTRL |= OSC_PLLEN_bm;
    while (!(OSC.STATUS & OSC_PLLRDY_bm));

    /* and switch the CPU core to run from it */
    CCP_write( &CLK.CTRL, CLK_SCLKSEL_PLL_gc );

    /* Wait for the system to switch clock */
    _delay_us(2);

    /* Select 32 kHz as external clock */
    OSC.XOSCCTRL = OSC_XOSCSEL_32KHz_gc;

    /* Start the 32MHz internal RC oscillator and external clock, but keep 2MHz clock
         on */
    OSC.CTRL |= OSC_XOSCEN_bm | OSC_RC32MEN_bm;

    /* Wait until 32MHz clock is ready */
    while( !(OSC.STATUS & OSC_RC32MRDY_bm) );

    /* Wait for the external oscillator to stabilize. */
    while ( !(OSC.STATUS & OSC_XOSCRDY_bm) );

    /* and start the DFLL to increase it to 48MHz using the USB SOF as a reference */
    OSC.DFLLCTRL    |= (2 << OSC_RC32MCREF_gp);
    DFLLRC32M.COMP1 = ((F_USB / 1000) & 0xFF);
    DFLLRC32M.COMP2 = ((F_USB / 1000) >> 8);

    NVM.CMD         = NVM_CMD_READ_CALIB_ROW_gc;
    DFLLRC32M.CALA = pgm_read_byte(offsetof(NVM_PROD_SIGNATURES_t, USBRCOSCA));
    DFLLRC32M.CALB = pgm_read_byte(offsetof(NVM_PROD_SIGNATURES_t, USBRCOSC));
    NVM.CMD         = 0;

    /* Enable automatic run-time calibration */
    DFLLRC32M.CTRL = DFLL_ENABLE_bm;
}

void
rtc_init(void) {

    /* Set 32 kHz from external 32kHz oscillator as clock source for RTC. */
    CLK.RTCCTRL = CLK_RTCSRC_TOSC32_gc | CLK_RTCEN_bm;

    /* Wait until RTC is not busy */
    while ( RTC.STATUS & RTC_SYNCBUSY_bm );

    /* Period register value. Must subtract 1, because zero value is counted.
     * Gives an overflow every 1/1024 second */
    RTC.PER = 32 - 1;

    /* Make sure COMP and CNT is 0. */
    RTC.COMP = 0;
    RTC.CNT = 0x0000;

    /* Divide by 1, so 32.768 kHz frequency */
    RTC.CTRL = RTC_PRESCALER_DIV1_gc;

    /* Enable overflow interrupt. */
    RTC.INTCTRL = RTC_OVFINTLVL_HI_gc | RTC_COMPINTLVL_OFF_gc;
}

void
int_init() {

    /* Enable high, medium and low level interrupt */
    PMIC.CTRL = PMIC_LOLVLEN_bm | PMIC_MEDLVLEN_bm | PMIC_HILVLEN_bm;

    /* Enable global interrupt */
    sei();
}

void
led_init() {

    /* Set Green LED-pin PC2 as output */
    //LED_PORT.DIRSET = LED_PIN;

    /* Invert the LED-pin, so the LED light when we set it high */
    LED_PORT.PIN2CTRL |= PORT_INVEN_bm;

    /* Set it to light as default, so we can toogle it later on */
    LED_PORT.OUTSET = LED_PIN;
}

void print_accel(mpudata_t *mpu) {
    /* Clear screen */
    printf("\x1b[H\x1b[2J");

    printf("\rCalibrate␣accelerometer␣");
    printf("X␣%4d|%4d|%4d␣␣␣␣Y␣%4d|%4d|%4d␣␣␣␣Z␣%4d|%4d|%4d",
            accel_cal.minVal[0], mpu->rawAccel[0], accel_cal.maxVal[0],
            accel_cal.minVal[1], mpu->rawAccel[1], accel_cal.maxVal[1],
            accel_cal.minVal[2], mpu->rawAccel[2], accel_cal.maxVal[2]);

    //fflush(stdout);
}

```

```
void print_mag(mpudata_t *mpu) {
    /* Clear screen */
    printf("\x1b[H\x1b[2J");

    printf("\rCalibrate␣magnetometer␣");
    printf("X␣%4d|%4d|%4d␣␣␣␣Y␣%4d|%4d|%4d␣␣␣␣Z␣%4d|%4d|%4d",
            mag_cal.minVal[0], mpu->rawMag[0], mag_cal.maxVal[0],
            mag_cal.minVal[1], mpu->rawMag[1], mag_cal.maxVal[1],
            mag_cal.minVal[2], mpu->rawMag[2], mag_cal.maxVal[2]);

    //fflush(stdout);
}

void print_fused_euler_angles_as_nmea(mpudata_t *mpu) {

    char nmea_sentence_time[80];
    char nmea_sentence_data[80];
    char nmea_checksum_buffer[3];
    char nmea_sentence[80];

    sprintf(nmea_sentence_time, "$PASHR,%0.2d%0.2d%0.2d.%0.3d,",
           relative_time.hours,
           relative_time.minutes,
           relative_time.secs,
           (relative_time.ticks*1000UL)/1024UL);

    sprintf(nmea_sentence_data, "%␣7.2f,M,%␣7.2f,%␣7.2f,,0.1,0.1,1.0,0,1",
            mpu->fusedEuler[VEC3_Z] * RAD_TO_DEGREE,
            mpu->fusedEuler[VEC3_X] * RAD_TO_DEGREE,
            mpu->fusedEuler[VEC3_Y] * RAD_TO_DEGREE);

    sprintf(nmea_sentence, "%s%s*", nmea_sentence_time, nmea_sentence_data);
    sprintf(nmea_checksum_buffer, "%02X", nmea_checksum(nmea_sentence));
    sprintf(nmea_sentence, "%s%s", nmea_sentence, nmea_checksum_buffer);
    printf("%s\r\n", nmea_sentence);
}


void print_fused_euler_angles(mpudata_t *mpu) {
    const number_of_samples = 50;

    //avarage_pan = ((number_of_samples - 1)*avarage_pan + mpu->fusedEuler[VEC3_Z] *
        RAD_TO_DEGREE)/number_of_samples;

    printf("\r\n%0.2d:%0.2d:%0.2d.%0.3d,␣",
            relative_time.hours,
            relative_time.minutes,
            relative_time.secs,
            (relative_time.ticks*1000UL)/1024UL);
    printf("%␣6.1f,␣%␣6.1f,␣%␣4.0f",
            mpu->fusedEuler[VEC3_X] * RAD_TO_DEGREE,
            mpu->fusedEuler[VEC3_Y] * RAD_TO_DEGREE,
            mpu->fusedEuler[VEC3_Z] * RAD_TO_DEGREE);
            //avarage_pan);
}

void print_debug_data(mpudata_t *mpu) {

    printf(",␣%␣4i,␣%␣4i,␣%␣4i",
            mpu->rawMag[VEC3_X],
            mpu->rawMag[VEC3_Y],
            mpu->rawMag[VEC3_Z]);
    printf(",␣%␣4i,␣%␣4i,␣%␣4i",
            mpu->calibratedMag[VEC3_X],
            mpu->calibratedMag[VEC3_Y],
            mpu->calibratedMag[VEC3_Z]);
}

void parse_command(char command) {

    /* Start sending angles */
    if (command == 's') {
        running = TRUE;
        RTC.CNT = 0x0000;
        TCC0.CNT = 0;
        relative_time.ticks = 0;
        relative_time.secs = 0;
        relative_time.minutes = 0;
        relative_time.hours = 0;
        relative_time.days = 0;
        LED_PORT.DIRSET = LED_PIN;
        calibrate_mag = 0;
        calibrate_accel = 0;

        /* Clear screen */
        printf("\x1b[H\x1b[2J");
    }

    /* Stop sending angles */
    if (command == 'S') {
        running = FALSE;
        calibrate_mag = FALSE;
        calibrate_accel = FALSE;
        LED_PORT.DIRCLR = LED_PIN;
    }

    /* Reset counter */
    if (command == 'r') {
        RTC.CNT = 0x0000;
        TCC0.CNT = 0;
        relative_time.ticks = 0;
        relative_time.secs = 0;
        relative_time.minutes = 0;
        relative_time.hours = 0;
        relative_time.days = 0;
        calibrate_mag = FALSE;
        calibrate_accel = FALSE;
    }

    /* Send one measurment */
    if (command == '1') {
        send_one_measurment = TRUE;
        calibrate_mag = FALSE;
        calibrate_accel = FALSE;
    }

    /* Calibrate mag */
    if (command == 'm') {
        calibrate_mag = TRUE;
        calibrate_accel = FALSE;
        //change = TRUE;
    }

    /* Calibrate accel */
    if (command == 'a') {
        calibrate_mag = FALSE;
        calibrate_accel = TRUE;
        //change = TRUE;
    }

    /* Reset mag */
    if (command == 'M') {
        mag_cal.calibrated = FALSE;

        for (uint8_t i = 0; i < 3; i++) {
            mag_cal.minVal[i] = INT16_MAX;//0x7fff; ??
            mag_cal.maxVal[i] = INT16_MIN;//0x8000; ??
        }
        //eeprom_write_block(&accel_cal, &EEaccel_cal, sizeof(accel_cal_st));
    }

    /* Reset accel */
    if (command == 'A') {
        accel_cal.calibrated = FALSE;

        for (uint8_t i = 0; i < 3; i++) {
            accel_cal.minVal[i] = INT16_MAX;//0x7fff; ??
            accel_cal.maxVal[i] = INT16_MIN;//0x8000; ??
        }

        //eeprom_write_block(&accel_cal, &EEaccel_cal, sizeof(accel_cal_st));
    }

    /* Run bootloader */
    if (command == 'b') {
        printf("Resetting␣and␣starting␣bootoader\r\n");
        _delay_ms(1500);

        /* Start a software reset */
        CCP_write( &RST.CTRL, RST_SWRST_bm );
    }


    /* Get debug info */
    if (command == 'd') {
        printf("%d␣<->␣%d\n%d␣<->␣%d\n%d␣<->␣%d\n",
                mag_cal.minVal[0], mag_cal.maxVal[0],
                mag_cal.minVal[1], mag_cal.maxVal[1],
                mag_cal.minVal[2], mag_cal.maxVal[2]);
        printf("%d␣:␣%d\n%d␣:␣%d\n%d␣:␣%d\n",
                mag_cal_or.range[0], mag_cal_or.offset[0],
                mag_cal_or.range[1], mag_cal_or.offset[1],
                mag_cal_or.range[2], mag_cal_or.offset[2]);
    }

    if (command == 'w') {

        if (calibrate_accel) {
            accel_cal.calibrated = TRUE;
            eeprom_write_block(&accel_cal, &EEaccel_cal, sizeof(accel_cal_st));
            /* Clear screen */
            printf("\x1b[H\x1b[2J");
            printf("accel_cal␣values␣written␣to␣EEPROM.");
        }

        if (calibrate_mag) {
            mag_cal.calibrated = TRUE;
            eeprom_write_block(&mag_cal, &EEmag_cal, sizeof(mag_cal_st));
            /* Clear screen */
            printf("\x1b[H\x1b[2J");
            printf("mag_cal␣values␣written␣to␣EEPROM.");
        }

        calibrate_mag = FALSE;
        calibrate_accel = FALSE;
        running = FALSE;
    }

    /* Read EEPROM calibration data */
    if (command == 'e') {
        eeprom_read_block(&accel_cal, &EEaccel_cal, sizeof(accel_cal_st));
        eeprom_read_block(&mag_cal, &EEmag_cal, sizeof(mag_cal_st));

        /* Clear screen */
        printf("\x1b[H\x1b[2J");
        printf("accel_cal:␣X␣%4d|%4d␣␣␣␣Y␣%4d|%4d␣␣␣␣Z␣%4d|%4d\r\n",
                accel_cal.minVal[0], accel_cal.maxVal[0],
                accel_cal.minVal[1], accel_cal.maxVal[1],
                accel_cal.minVal[2], accel_cal.maxVal[2]);

        printf("mag_cal:␣␣␣X␣%4d|%4d␣␣␣␣Y␣%4d|%4d␣␣␣␣Z␣%4d|%4d",
                mag_cal.minVal[0], mag_cal.maxVal[0],
                mag_cal.minVal[1], mag_cal.maxVal[1],
                mag_cal.minVal[2], mag_cal.maxVal[2]);

        calibrate_mag = FALSE;
        calibrate_accel = FALSE;
        running = FALSE;
    }

    /* Read EEPROM serial number */
    if (command == 'i') {
        eeprom_read_block(&serial_number, &EEserial_number, sizeof(serial_number));
        printf("serial_number: %s\r\n", serial_number);
    }

#if 0
    /* Write EEPROM serial number */
    if (command == 'I') {
        eeprom_write_block(&serial_number, &EEserial_number, sizeof(serial_number));
    }
#endif
}

void run_bootloader_on_soft_reset() {
    if (RST.STATUS & RST_SRF_bm) {

        LED_PORT.DIRSET = LED_PIN;

        for(uint8_t i; i < 7; i++) {
            LED_PORT.OUTTGL = LED_PIN;
            _delay_ms(200);
        }

        cli();
        EIND = BOOT_SECTION_START>>17;
        start_bootloader();
    }
}

void load_calibration_from_eeprom() {
    eeprom_read_block(&accel_cal, &EEaccel_cal, sizeof(accel_cal_st));
    eeprom_read_block(&mag_cal, &EEmag_cal, sizeof(mag_cal_st));


    accel_cal_or.offset[0] = (short) ((accel_cal.minVal[0] + accel_cal.maxVal[0]) / 2);
    accel_cal_or.offset[1] = (short) ((accel_cal.minVal[1] + accel_cal.maxVal[1]) / 2);
    accel_cal_or.offset[2] = (short) ((accel_cal.minVal[2] + accel_cal.maxVal[2]) / 2);

    accel_cal_or.range[0] = (short) (accel_cal.maxVal[0] - accel_cal_or.offset[0]);
    accel_cal_or.range[1] = (short) (accel_cal.maxVal[1] - accel_cal_or.offset[1]);
    accel_cal_or.range[2] = (short) (accel_cal.maxVal[2] - accel_cal_or.offset[2]);


    mag_cal_or.offset[0] = (short) ((mag_cal.minVal[0] + mag_cal.maxVal[0]) / 2);
    mag_cal_or.offset[1] = (short) ((mag_cal.minVal[1] + mag_cal.maxVal[1]) / 2);
    mag_cal_or.offset[2] = (short) ((mag_cal.minVal[2] + mag_cal.maxVal[2]) / 2);

    mag_cal_or.range[0] = (short) (mag_cal.maxVal[0] - mag_cal_or.offset[0]);
    mag_cal_or.range[1] = (short) (mag_cal.maxVal[1] - mag_cal_or.offset[1]);
    mag_cal_or.range[2] = (short) (mag_cal.maxVal[2] - mag_cal_or.offset[2]);

    if (accel_cal.calibrated == TRUE) {
        mpu9150_set_accel_cal(&accel_cal_or);
    }

    if (mag_cal.calibrated == TRUE) {
        mpu9150_set_mag_cal(&mag_cal_or);
    }
}


int
main() {
    clock_init();
    led_init();
    run_bootloader_on_soft_reset();
    //uart_init();
    rtc_init();
    twi_init();
    int_init();
    usb_talk_init();

    /* Hack: set the serial_number */
    eeprom_write_block(&serial_number, &EEserial_number, sizeof(serial_number));

    _delay_ms(50);

    mpu9150_init(SAMPLE_RATE, YAW_MIXING_FACTOR);

    load_calibration_from_eeprom();

    mpudata_t mpu;

    memset(&mpu, 0, sizeof(mpudata_t));

    for (uint8_t i = 0; i < 3; i++) {
        accel_cal.minVal[i] = INT16_MAX; //0x7fff; ??
        accel_cal.maxVal[i] = INT16_MIN; //0x8000; ??
    }

    for (uint8_t i = 0; i < 3; i++) {
        mag_cal.minVal[i] = INT16_MAX; //0x7fff; ??
        mag_cal.maxVal[i] = INT16_MIN; //0x8000; ??
    }

    for(;;) {

        /* Receive bytes from the host */
        char command = CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);
        parse_command(command);

        CDC_Device_USBTask(&VirtualSerial_CDC_Interface);

        USB_USBTask();

        if (mpu9150_read(&mpu) == 0) {
            if (calibrate_mag) {
                for (uint8_t i = 0; i < 3; i++) {
                    if (mpu.rawMag[i] < mag_cal.minVal[i]) {
                        mag_cal.minVal[i] = mpu.rawMag[i];
                        //change = TRUE;
                    }

                    if (mpu.rawMag[i] > mag_cal.maxVal[i]) {
                        mag_cal.maxVal[i] = mpu.rawMag[i];
                        //change = TRUE;
                    }
                }
            } else if (calibrate_accel) {
                for (uint8_t i = 0; i < 3; i++) {
                    if (mpu.rawAccel[i] < accel_cal.minVal[i]) {
                        accel_cal.minVal[i] = mpu.rawAccel[i];
                        //change = TRUE;
                    }

                    if (mpu.rawAccel[i] > accel_cal.maxVal[i]) {
                        accel_cal.maxVal[i] = mpu.rawAccel[i];
                        //change = TRUE;
                    }
                }
            } else {
                if (send_one_measurment == TRUE) {
                    print_fused_euler_angles_as_nmea(&mpu);
                    //print_fused_euler_angles(&mpu);
                    //print_debug_data(&mpu);
                    send_one_measurment = FALSE;

                } else if (running) {
                    print_fused_euler_angles_as_nmea(&mpu);
                    //print_debug_data(&mpu);
                }
            }
        }

        //if (change) {
            if (calibrate_mag)
                print_mag(&mpu);
            else if (calibrate_accel)
                print_accel(&mpu);
        //}

        //change = FALSE;

        // 50 Hertz
        _delay_ms(20);
    }
}

ISR(RTC_OVF_vect) {

    /* Ticks is 1/1024 seconds */
    relative_time.ticks++;

    if (relative_time.ticks > 1023) {
        relative_time.ticks = 0;
        relative_time.secs++;

        LED_PORT.OUTTGL = LED_PIN;
    }

    if (relative_time.secs > 59) {
        relative_time.secs = 0;
        relative_time.minutes++;
    }

    if (relative_time.minutes > 59) {
        relative_time.minutes = 0;
        relative_time.hours++;
    }

    if (relative_time.hours > 23) {
        relative_time.hours = 0;
        relative_time.days++;
    }
}
```

code/hydroAHRS_mkII/main.h

```
# ifndef MAIN_H
# define MAIN_H

#define TRUE 1
#define FALSE 0

#define SAMPLE_RATE         50
#define YAW_MIXING_FACTOR 4

#define LOOP_DELAY ((1000 / SAMPLE_RATE) - 2)

#define LENGTH(array) (sizeof(array) / sizeof(array[0]))

#define delay_ms     _delay_ms
#define get_ms       linux_get_ms
#define log_i        printf
#define log_e        printf
#define min(a, b)    ((a < b) ? a : b)

static inline int reg_int_cb(struct int_param_s *int_param) {
    return 0;
}

#define MAX_CMD_LENGTH 20

typedef struct {
    uint16_t ticks;
    uint8_t secs;
    uint8_t minutes;
    uint8_t hours;
    uint16_t days;

} relative_time_t;

void __no_operation(void);

extern relative_time_t relative_time;
extern uint8_t running;
extern uint8_t send_one_measurment;

#endif
```

code/hydroAHRS_mkII/mpu9150.c

```
////////////////////////////////////////////////////////////////////////////
//
//  This file is part of linux-mpu9150
//
//  Copyright (c) 2013 Pansenti, LLC
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy of
//  this software and associated documentation files (the "Software"), to deal in
//  the Software without restriction, including without limitation the rights to use,
//  copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
//  Software, and to permit persons to whom the Software is furnished to do so,
//  subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
//  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
//  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
//  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
//  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#include <inttypes.h>

#include "eMPL/inv_mpu.h"
#include "eMPL/inv_mpu_dmp_motion_driver.h"
#include "mpu9150.h"

static int data_ready();
static void calibrate_data(mpudata_t *mpu);
static void tilt_compensate(quaternion_t magQ, quaternion_t unfusedQ);
static int data_fusion(mpudata_t *mpu);
static unsigned short inv_row_2_scale(const signed char *row);
static unsigned short inv_orientation_matrix_to_scalar(const signed char *mtx);

int debug_on;
int yaw_mixing_factor;

#if 0
int use_accel_cal;
caldata_t accel_cal_data;

int use_mag_cal;
caldata_t mag_cal_data;
#endif

void mpu9150_set_debug(int on) {
    debug_on = on;
}

uint8_t mpu9150_init(uint8_t sample_rate, uint8_t yaw_mixing_factor) {


    signed char gyro_orientation[9] = { 1, 0, 0,
                                        0, 1, 0,
                                         0, 0, 1 };

    mpu_init();
    mpu_set_sensors(INV_XYZ_GYRO | INV_XYZ_ACCEL | INV_XYZ_COMPASS);
    mpu_configure_fifo(INV_XYZ_GYRO | INV_XYZ_ACCEL);
    mpu_set_sample_rate(sample_rate);
    mpu_set_compass_sample_rate(sample_rate);
    dmp_load_motion_driver_firmware();
    dmp_set_orientation(inv_orientation_matrix_to_scalar(gyro_orientation));
    dmp_enable_feature(DMP_FEATURE_6X_LP_QUAT | DMP_FEATURE_SEND_RAW_ACCEL |
            DMP_FEATURE_SEND_CAL_GYRO | DMP_FEATURE_GYRO_CAL);
    dmp_set_fifo_rate(sample_rate);
    mpu_set_dmp_state(1);

    return 0;
}

void mpu9150_exit() {
    // Turn off the DMP on exit
    mpu_set_dmp_state(0);

    // TODO: Should turn off the sensors too
}

#if 0
void mpu9150_set_accel_cal(caldata_t *cal) {
    int i;
    long bias[3];

    if (!cal) {
        use_accel_cal = 0;
        return;
    }

    memcpy(&accel_cal_data, cal, sizeof(caldata_t));

    for (i = 0; i < 3; i++) {
        if (accel_cal_data.range[i] < 1)
            accel_cal_data.range[i] = 1;
        else if (accel_cal_data.range[i] > ACCEL_SENSOR_RANGE)
            accel_cal_data.range[i] = ACCEL_SENSOR_RANGE;

        bias[i] = -accel_cal_data.offset[i];
    }

    if (debug_on) {
        printf("\naccel␣cal␣(range␣:␣offset)\n");

        //for (i = 0; i < 3; i++)
            //printf("%d : %d\n", accel_cal_data.range[i], accel_cal_data.offset[i]);
    }

    mpu_set_accel_bias(bias);

    use_accel_cal = 1;
}

void mpu9150_set_mag_cal(caldata_t *cal) {
    int i;

    if (!cal) {
        use_mag_cal = 0;
        return;
    }

    memcpy(&mag_cal_data, cal, sizeof(caldata_t));

    for (i = 0; i < 3; i++) {
        if (mag_cal_data.range[i] < 1)
            mag_cal_data.range[i] = 1;
        else if (mag_cal_data.range[i] > MAG_SENSOR_RANGE)
            mag_cal_data.range[i] = MAG_SENSOR_RANGE;

        if (mag_cal_data.offset[i] < -MAG_SENSOR_RANGE)
            mag_cal_data.offset[i] = -MAG_SENSOR_RANGE;
        else if (mag_cal_data.offset[i] > MAG_SENSOR_RANGE)
            mag_cal_data.offset[i] = MAG_SENSOR_RANGE;
    }

    if (1) {
        printf("\nmag␣cal␣(range␣:␣offset)\n");

        for (i = 0; i < 3; i++)
            printf("%d␣:␣%d\n", mag_cal_data.range[i], mag_cal_data.offset[i]);
    }

    use_mag_cal = 1;
}
#endif

int mpu9150_read_dmp(mpudata_t *mpu) {

    short sensors;
    unsigned char more;


    if (!data_ready())
        return -1;

    if (dmp_read_fifo(mpu->rawGyro, mpu->rawAccel, mpu->rawQuat, &mpu->dmpTimestamp, &sensors, &more) < 0) {
        printf("dmp_read_fifo()␣failed\n");
        return -1;
    }

    while (more) {
        // Fell behind, reading again
        if (dmp_read_fifo(mpu->rawGyro, mpu->rawAccel, mpu->rawQuat, &mpu->dmpTimestamp, &sensors, &more) < 0) {
            printf("dmp_read_fifo()␣failed\n");
            return -1;
        }
    }

    return 0;
}

int mpu9150_read_mag(mpudata_t *mpu) {

    if (mpu_get_compass_reg(mpu->rawMag, 0) < 0) {
        printf("mpu_get_compass_reg()␣failed\n");
        return -1;
    }

    return 0;
}

int mpu9150_read(mpudata_t *mpu) {

    if (mpu9150_read_dmp(mpu) != 0)
        return -1;

    if (mpu9150_read_mag(mpu) != 0)
        return -1;

    calibrate_data(mpu);

    return data_fusion(mpu);
}

int data_ready() {

    short status;

    if (mpu_get_int_status(&status) < 0) {
        printf("mpu_get_int_status()␣failed\n");
        return 0;
    }

    // debug
    //if (status != 0x0103)
    //  fprintf(stderr, "%04X\n", status);

    return (status == (MPU_INT_STATUS_DATA_READY | MPU_INT_STATUS_DMP |
        MPU_INT_STATUS_DMP_0));
}

void calibrate_data(mpudata_t *mpu) {

    if (use_mag_cal) {
      mpu->calibratedMag[VEC3_Y] = -(short)(((long)(mpu->rawMag[VEC3_X] - mag_cal_data.
            offset[VEC3_X])
              * (long)MAG_SENSOR_RANGE) / (long)mag_cal_data.range[VEC3_X]);

      mpu->calibratedMag[VEC3_X] = (short)(((long)(mpu->rawMag[VEC3_Y] - mag_cal_data.
            offset[VEC3_Y])
              * (long)MAG_SENSOR_RANGE) / (long)mag_cal_data.range[VEC3_Y]);

      mpu->calibratedMag[VEC3_Z] = (short)(((long)(mpu->rawMag[VEC3_Z] - mag_cal_data.
            offset[VEC3_Z])
              * (long)MAG_SENSOR_RANGE) / (long)mag_cal_data.range[VEC3_Z]);
    } else {
        mpu->calibratedMag[VEC3_X] = mpu->rawMag[VEC3_Y];
        mpu->calibratedMag[VEC3_Y] = -mpu->rawMag[VEC3_X];
        mpu->calibratedMag[VEC3_Z] = mpu->rawMag[VEC3_Z];
    }

    if (use_accel_cal) {
      mpu->calibratedAccel[VEC3_X] = -(short)(((long)mpu->rawAccel[VEC3_X] * (long)
           ACCEL_SENSOR_RANGE)
              / (long)accel_cal_data.range[VEC3_X]);

      mpu->calibratedAccel[VEC3_Y] = (short)(((long)mpu->rawAccel[VEC3_Y] * (long)
           ACCEL_SENSOR_RANGE)
              / (long)accel_cal_data.range[VEC3_Y]);

      mpu->calibratedAccel[VEC3_Z] = (short)(((long)mpu->rawAccel[VEC3_Z] * (long)
           ACCEL_SENSOR_RANGE)
              / (long)accel_cal_data.range[VEC3_Z]);
    } else {
        mpu->calibratedAccel[VEC3_X] = -mpu->rawAccel[VEC3_X];
        mpu->calibratedAccel[VEC3_Y] = mpu->rawAccel[VEC3_Y];
        mpu->calibratedAccel[VEC3_Z] = mpu->rawAccel[VEC3_Z];
    }
}

void tilt_compensate(quaternion_t magQ, quaternion_t unfusedQ) {
    quaternion_t unfusedConjugateQ;
    quaternion_t tempQ;

    quaternionConjugate(unfusedQ, unfusedConjugateQ);
    quaternionMultiply(magQ, unfusedConjugateQ, tempQ);
    quaternionMultiply(unfusedQ, tempQ, magQ);
}

int data_fusion(mpudata_t *mpu) {
    quaternion_t dmpQuat;
    vector3d_t dmpEuler;
    quaternion_t magQuat;
    quaternion_t unfusedQuat;
    float deltaDMPYaw;
    float deltaMagYaw;
    float newMagYaw;
    float newYaw;

    dmpQuat[QUAT_W] = (float)mpu->rawQuat[QUAT_W];
    dmpQuat[QUAT_X] = (float)mpu->rawQuat[QUAT_X];
    dmpQuat[QUAT_Y] = (float)mpu->rawQuat[QUAT_Y];
    dmpQuat[QUAT_Z] = (float)mpu->rawQuat[QUAT_Z];

    quaternionNormalize(dmpQuat);
    quaternionToEuler(dmpQuat, dmpEuler);

    mpu->dmpEuler[VEC3_X] = dmpEuler[VEC3_X];
    mpu->dmpEuler[VEC3_Y] = dmpEuler[VEC3_Y];
    mpu->dmpEuler[VEC3_Z] = dmpEuler[VEC3_Z];

    mpu->fusedEuler[VEC3_X] = dmpEuler[VEC3_X];
    mpu->fusedEuler[VEC3_Y] = -dmpEuler[VEC3_Y];
    mpu->fusedEuler[VEC3_Z] = 0;

    eulerToQuaternion(mpu->fusedEuler, unfusedQuat);

    // Maybe the drift appeared here?
    deltaDMPYaw = -dmpEuler[VEC3_Z] + mpu->lastDMPYaw;
    //if ((deltaDMPYaw <= 0.0001) || (-deltaDMPYaw <= 0.0001)) {
    if (fabs(deltaDMPYaw) < 0.0001) {
        deltaDMPYaw = 0;
    }
    //printf("deltaDMPYaw(% 6.10f) = -dmpEuler[VEC3_Z](% 6.10f) + mpu->lastDMPYaw(%
         6.10f)\n\r", deltaDMPYaw, dmpEuler[VEC3_Z], mpu->lastDMPYaw);
    mpu->lastDMPYaw = dmpEuler[VEC3_Z];

    magQuat[QUAT_W] = 0;
    magQuat[QUAT_X] = mpu->calibratedMag[VEC3_X];
    magQuat[QUAT_Y] = mpu->calibratedMag[VEC3_Y];
    magQuat[QUAT_Z] = mpu->calibratedMag[VEC3_Z];

    tilt_compensate(magQuat, unfusedQuat);

    newMagYaw = -atan2f(magQuat[QUAT_Y], magQuat[QUAT_X]);

    if (newMagYaw != newMagYaw) {
        printf("newMagYaw␣NAN\n");
        return -1;
    }

    if (newMagYaw < 0.0f)
        newMagYaw = TWO_PI + newMagYaw;

    newYaw = mpu->lastYaw + deltaDMPYaw;

    if (newYaw > TWO_PI)
        newYaw -= TWO_PI;
    else if (newYaw < 0.0f)
        newYaw += TWO_PI;

    deltaMagYaw = newMagYaw - newYaw;

    if (deltaMagYaw >= (float)M_PI)
        deltaMagYaw -= TWO_PI;
    else if (deltaMagYaw < -(float)M_PI)
        deltaMagYaw += TWO_PI;

    //<yaw-mix-factor>   Effect of mag yaw on fused yaw data.
    //                   0 = gyro only
    //                   1 = mag only
    //                   > 1 scaled mag adjustment of gyro data
    //                   The default is 4.
    if (yaw_mixing_factor > 0)
        newYaw += deltaMagYaw / yaw_mixing_factor;

    if (newYaw > TWO_PI)
        newYaw -= TWO_PI;
    else if (newYaw < 0.0f)
        newYaw += TWO_PI;

    mpu->lastYaw = newYaw;

    if (newYaw > (float)M_PI)
        newYaw -= TWO_PI;

    //mpu->fusedEuler[VEC3_Z] = newYaw;
    //mpu->fusedEuler[VEC3_Z] = newMagYaw;
    mpu->fusedEuler[VEC3_Z] = dmpEuler[VEC3_Z];

    eulerToQuaternion(mpu->fusedEuler, mpu->fusedQuat);

    return 0;
}

/* These next two functions convert the orientation matrix (see
 * gyro_orientation) to a scalar representation for use by the DMP.
 * NOTE: These functions are borrowed from InvenSense's MPL.
 */
unsigned short inv_row_2_scale(const signed char *row) {

    unsigned short b;

    if (row[0] > 0)
        b = 0;
    else if (row[0] < 0)
        b = 4;
    else if (row[1] > 0)
        b = 1;
    else if (row[1] < 0)
        b = 5;
    else if (row[2] > 0)
        b = 2;
    else if (row[2] < 0)
        b = 6;
    else
        b = 7;      // error
    return b;
}

unsigned short inv_orientation_matrix_to_scalar(const signed char *mtx) {

    unsigned short scalar;
    /*
       XYZ  010_001_000 Identity Matrix
       XZY  001_010_000
       YXZ  010_000_001
       YZX  000_010_001
       ZXY  001_000_010
       ZYX  000_001_010
     */
    scalar = inv_row_2_scale(mtx);
    scalar |= inv_row_2_scale(mtx + 3) << 3;
    scalar |= inv_row_2_scale(mtx + 6) << 6;
    return scalar;
}
```

code/hydroAHRS_mkII/mpu9150.h

```
////////////////////////////////////////////////////////////////////////////
//
//  This file is part of linux-mpu9150
//
//  Copyright (c) 2013 Pansenti, LLC
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy of
//  this software and associated documentation files (the "Software"), to deal in
//  the Software without restriction, including without limitation the rights to use,
//  copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
//  Software, and to permit persons to whom the Software is furnished to do so,
//  subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
//  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
//  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
//  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
//  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#ifndef MPU9150_H
#define MPU9150_H

#include "quaternion.h"

#define MAG_SENSOR_RANGE     4096
#define ACCEL_SENSOR_RANGE   32000

// Somewhat arbitrary limits here. The values are samples per second.
// The MIN comes from the way we are timing our loop in imu and imucal.
// That's easily worked around, but no one probably cares.
// The MAX comes from the compass. This could be avoided with separate
// sample rates for the compass and the accel/gyros which can handle
// faster sampling rates. This is a TODO item to see if it's useful.
// There are some practical limits on the speed that come from a 'userland'
// implementation like this as opposed to a kernel or 'bare-metal' driver.
#define MIN_SAMPLE_RATE 2
#define MAX_SAMPLE_RATE 100

typedef struct {
    short offset[3];
    short range[3];
} caldata_t;

typedef struct {
    short rawGyro[3];
    short rawAccel[3];
    long rawQuat[4];
    unsigned long dmpTimestamp;

    short rawMag[3];
    unsigned long magTimestamp;

    short calibratedAccel[3];
    short calibratedMag[3];

    quaternion_t fusedQuat;
    vector3d_t fusedEuler;

    vector3d_t dmpEuler;

    float lastDMPYaw;
    float lastYaw;
} mpudata_t;


void mpu9150_set_debug(int on);
uint8_t mpu9150_init(uint8_t sample_rate, uint8_t yaw_mixing_factor);
void mpu9150_exit();
int mpu9150_read(mpudata_t *mpu);
int mpu9150_read_dmp(mpudata_t *mpu);
int mpu9150_read_mag(mpudata_t *mpu);
void mpu9150_set_accel_cal(caldata_t *cal);
void mpu9150_set_mag_cal(caldata_t *cal);

#endif
```

code/hydroAHRS_mkII/nmea.c

```
#include "nmea.h"


#if 0
void handle_gnss_data() {
    get_uart_str(GNSS_DEVICE, message[GNSS_DEVICE]);

    /* Get the wanted message type */
    if(!strncmp(message[GNSS_DEVICE], "$GPRMC", 6)) {
        printf("%s\r\n", message[GNSS_DEVICE]);
        //nmea_checksum(&message[GNSS_DEVICE]);
    }
}
#endif

uint8_t nmea_checksum(int8_t *nmea_sentence) {

    //char checksum[3];

    uint8_t temp = 0;

    // 1 because we want to skip the $
    uint8_t i = 1;

    //while(nmea_sentence[i++] != ',');

    /* Calculate checksum */
    while(nmea_sentence[i] != '*')

          // XOR each character
          temp ^= nmea_sentence[i++];

    return temp;

    //sprintf(checksum, "%02X\n", temp);

#if 0
    /* Compare the two last characters in nmea_sentence with the checksum in (ascii) */
    if (nmea_sentence[++i] == checksum[0] &&
        nmea_sentence[++i] == checksum[1]) {

        return 0;

    } else {

        return 1;
    }
#endif
}

uint8_t nmea_parse(int8_t *nmea_sentence) {

    printf("nmea: %s\r\n", nmea_sentence);

    return 0;
}
```

code/hydroAHRS_mkII/nmea.h

```
#ifndef NMEA_H
#define NMEA_H

#include <inttypes.h>

uint8_t nmea_checksum(int8_t *nmea_sentence);
uint8_t nmea_parse(int8_t *nmea_sentence);

#endif
```

code/hydroAHRS_mkII/twi.c

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <inttypes.h>

#include "main.h"
#include "uart.h"
#include "twi.h"
#include "twim.h"
```

```

void
twi_init() {
    /* Configure SDA as output */
    PORTC.DIRSET = PIN0_bm;

    /* Configure SCL as output */
    PORTC.DIRSET = PIN1_bm;

    /* Enable TWI Master and (high level) TWI read and write interrupt */
    //TWIC.MASTER.CTRLA = TWI_MASTER_ENABLE_bm | TWI_MASTER_RIEN_bm | TWI_MASTER_WIEN_bm;
    //TWIC.MASTER.CTRLA |= TWI_MASTER_INTLVL0_bm | TWI_MASTER_INTLVL1_bm;

    /* Set the TWI frequency */
    //TWIC_MASTER_BAUD = TWI_FREQ;

    /* Force TWI bus to idle state */
    //TWIC.MASTER.STATUS = TWI_MASTER_BUSSTATE0_bm;

    twi_options_t m_options = {
        .speed = TWI_SPEED,
        //.chip  = TWI_MASTER_ADDR,
        .chip  = 0x00,
        .speed_reg = TWI_FREQ
    };

    twi_master_init(&TWI_MASTER, &m_options);
    twi_master_enable(&TWI_MASTER);
}

int
i2c_write(unsigned char slave_addr, unsigned char reg_addr,
        unsigned char length, unsigned char const *data) {


    uint8_t tmp[20];

    status_code_t master_status;

    tmp[0] = reg_addr;

    for (uint8_t i = 0; i < length; i++) {
        tmp[i+1] = data[i];
    }

    // Package to send
    twi_package_t packet = {
        .addr_length = 0,
        .chip        = slave_addr,
        .buffer      = (void *)tmp,
        .length      = length + 1,
        .no_wait     = false
    };

#if DEBUG
    printf("\ti2c_write(%02X,␣%02X,␣%u,␣[", slave_addr, reg_addr, length);

    if (length == 0) {
        printf("␣NULL␣]␣)\n");
    }
    else {
        for (uint8_t i = 0; i < length; i++)
            printf("␣%02X", data[i]);

    printf("␣]␣)\n");
    }
#endif

    master_status = twi_master_write(&TWI_MASTER, &packet);
}

int
```

```
i2c_read(unsigned char slave_addr, unsigned char reg_addr,
      unsigned char length, unsigned char *data) {

    status_code_t master_status;

#if DEBUG
    printf("\ti2c_read(%02X,␣%02X,␣%u,␣...)\n", slave_addr, reg_addr, length);
#endif

    uint8_t tmp[2];
    tmp[0] = reg_addr;
    twi_package_t packet = {
        .addr_length = 0,
        .chip        = slave_addr,
        .buffer      = tmp,
        .length      = 1,
        .no_wait     = false
    };

    master_status = twi_master_write(&TWI_MASTER, &packet);

    twi_package_t packet_rcv = {
        .addr_length = 0,
        .chip        = slave_addr,
        .buffer      = data,
        .length      = length,
        .no_wait     = false
    };

    master_status = twi_master_read(&TWI_MASTER, &packet_rcv);

#if DEBUG
    printf("\tLeaving␣i2c_read(),␣read␣%d␣bytes:␣", length);

    for (uint8_t i = 0; i < length; i++)
        printf("%02X␣", data[i]);

    printf("\n");
#endif

    return 0;
}
```

code/hydroAHRS_mkII/twi.h

```
# ifndef TWI_H
# define TWI_H

#define TWI_MASTER            TWIC
#define TWI_MASTER_PORT       PORTC
#define TWI_SPEED             400000

//#define TWI_FREQ ((F_CPU / (2.0 * (400000))) - 5)
#define TWI_FREQ ((F_CPU / (2.0 * (TWI_SPEED))) - 5)

//#define NORMAL  0  // Normal successful data transfer
//#define ARBLOST 1  // Arbitration lost condition
//#define BUSERR  2  // Bus error condition
//#define NEGACK  3  // Receive NACK condition

void twi_init(void);
//void send_cmd(uint8_t addr, uint8_t value);
uint8_t read_reg(uint8_t addr);
void write_reg(uint8_t addr, uint8_t value);

extern int8_t rxBuf[];
extern uint8_t i2c_slave_addr;

#endif
```

code/hydroAHRS_mkII/uart.c

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

#include "uart.h"

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

void
uart_init(void) {

    /* Set PD3 (TX) as output */
    PORTD.DIRSET = PIN3_bm;

    /* Set the baud rate */
    // USARTC1.BAUDCTRLA = (uint8_t) BAUD;
    // USARTC1.BAUDCTRLB = ((USARTC1.BAUDCTRLB) & 0xF0) | ((BAUD >> 8) & 0x0F);
    USARTD0.BAUDCTRLA = 0x2E;
    USARTD0.BAUDCTRLB = 0x98;

    /* Enable low level UART receive interrupt */
    USARTD0.CTRLA = USART_RXCINTLVL_LO_gc;

    /* Enable the UART transmitter and receiver */
    USARTD0.CTRLB = USART_RXEN_bm | USART_TXEN_bm;

    /* Asynchronous USART, no parity, 1 stop bit, 8 data bits */
    USARTD0.CTRLC = USART_CHSIZE0_bm | USART_CHSIZE1_bm;

    //stdout = &mystdout;
}

void
uart_send_byte(uint8_t ch) {

    /* Wait for TX buffer to be empty,
     * which is indicated by the UDRE1 bit being cleared */
    while (!(USARTD0.STATUS & USART_DREIF_bm)) {
    }

    USARTD0.DATA = ch;
}

void
uart_send_hex(uint8_t ch) {
    uint8_t temp;

    uart_send_str("0x");

    /* Send high nibble */
    temp = (uint8_t) ((ch & 0xF0) >> 4 ) + 0x30;

    if (temp > 0x39)
        temp += 7;

    uart_send_byte(temp);

    /* Send low nibble */
    temp = (ch & 0x0F) + 0x30;

    if (temp > 0x39)
        temp += 7;

    uart_send_byte(temp);
}

void
uart_send_str(char *ptr) {

    /* Send the string, one byte at time */
    while(*ptr) {
        uart_send_byte(*ptr);
        ptr++;
    }
}

void
uart_send_newline(void) {

    /* Send newline and return */
    uart_send_byte('\n');
    uart_send_byte('\r');
}


int uart_putchar (char ch, FILE *stream) {
    if (ch == '\n')
        uart_putchar('\r', stream);

    uart_send_byte(ch);

    return 0;
}
```

code/hydroAHRS_mkII/uart.h

```
#ifndef UART_H
#define UART_H

#define BAUD ((uint16_t) ((F_CPU / (16.0 * (115200))) + 0.5) - 1)

void uart_init(void);
void uart_send_byte(uint8_t ch);
void uart_send_hex(uint8_t ch);
void uart_send_str(char *ptr);
void uart_send_newline(void);
int uart_putchar(char c, FILE *stream);

#endif
```

code/hydroAHRS_mkII/usb_talk.c

```
#include "usb_talk.h"

/*  LUFA CDC Class driver interface configuration and state information. This
 *  structure is passed to all CDC Class driver functions, so that multiple
 *  instances of the same class within a device can be differentiated from one
 *  another.
 */
USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface = {
    .Config = {
        .ControlInterfaceNumber   = 0,
        .DataINEndpoint           = {
            .Address          = CDC_TX_EPADDR,
            .Size             = CDC_TXRX_EPSIZE,
            .Banks            = 1,
        },
        .DataOUTEndpoint = {
            .Address          = CDC_RX_EPADDR,
            .Size             = CDC_TXRX_EPSIZE,
            .Banks            = 1,
        },
        .NotificationEndpoint = {
            .Address          = CDC_NOTIFICATION_EPADDR,
            .Size             = CDC_NOTIFICATION_EPSIZE,
            .Banks            = 1,
        },
    },
};

void usb_talk_init(void) {
    USB_Init();

    /* Create a regular character stream for the interface so that it can be
     * used with the stdio.h functions */
    CDC_Device_CreateStream(&VirtualSerial_CDC_Interface, stdout);

    //LEDs_SetAllLEDs(LEDMASK_USB_NOTREADY);
}

#if 0
void usb_talk_task(void) {
        /* Receive bytes from the host */
        char c = CDC_Device_ReceiveByte(&VirtualSerial_CDC_Interface);

        switch (c) {
            case 'b':
                printf("Starting bootloader...\r\n");

#if 0
                cli();
                _delay_ms(2000);
                EIND = BOOT_SECTION_START>>17;
                start_bootloader();
#endif
                break;
        }


        CDC_Device_USBTask(&VirtualSerial_CDC_Interface);

        USB_USBTask();
}
#endif

void EVENT_USB_Device_Connect(void) {
    //LEDs_SetAllLEDs(LEDMASK_USB_ENUMERATING);
}

void EVENT_USB_Device_Disconnect(void) {
    //LEDs_SetAllLEDs(LEDMASK_USB_NOTREADY);
}

void EVENT_USB_Device_ConfigurationChanged(void) {
    bool ConfigSuccess = true;

    ConfigSuccess &= CDC_Device_ConfigureEndpoints(&VirtualSerial_CDC_Interface);

    //LEDs_SetAllLEDs(ConfigSuccess ? LEDMASK_USB_READY : LEDMASK_USB_ERROR);
}

void EVENT_USB_Device_ControlRequest(void) {
    CDC_Device_ProcessControlRequest(&VirtualSerial_CDC_Interface);
}
```

code/hydroAHRS_mkII/usb_talk.h

```
#ifndef _USB_TALK_H_
#define _USB_TALK_H_

#include "Descriptors.h"

#include <LUFA/Drivers/USB/USB.h>

void usb_talk_init(void);
void usb_talk_task(void);

void EVENT_USB_Device_Connect(void);
void EVENT_USB_Device_Disconnect(void);
void EVENT_USB_Device_ConfigurationChanged(void);
void EVENT_USB_Device_ControlRequest(void);

extern USB_ClassInfo_CDC_Device_t VirtualSerial_CDC_Interface;

#endif
```

code/hydroAHRS_mkII/vector3d.c

```c
////////////////////////////////////////////////////////////////////////////
//
//  This file is part of linux-mpu9150
//
//  Copyright (c) 2013 Pansenti, LLC
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy of
//  this software and associated documentation files (the "Software"), to deal in
//  the Software without restriction, including without limitation the rights to use,
//  copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
//  Software, and to permit persons to whom the Software is furnished to do so,
//  subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
//  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
//  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
//  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
//  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#include "vector3d.h"

void vector3DotProduct(vector3d_t a, vector3d_t b, float *d) {
    *d = a[VEC3_X] * b[VEC3_X] + a[VEC3_Y] * b[VEC3_Y] + a[VEC3_Z] * b[VEC3_Z];
}

void vector3CrossProduct(vector3d_t a, vector3d_t b, vector3d_t d) {
    d[VEC3_X] = a[VEC3_Y] * b[VEC3_Z] - a[VEC3_Z] * b[VEC3_Y];
    d[VEC3_Y] = a[VEC3_Z] * b[VEC3_X] - a[VEC3_X] * b[VEC3_Z];
    d[VEC3_Z] = a[VEC3_X] * b[VEC3_Y] - a[VEC3_Y] * b[VEC3_X];
}
```

code/hydroAHRS_mkII/vector3d.h

```c
////////////////////////////////////////////////////////////////////////////
//
//  This file is part of linux-mpu9150
//
//  Copyright (c) 2013 Pansenti, LLC
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy of
//  this software and associated documentation files (the "Software"), to deal in
//  the Software without restriction, including without limitation the rights to use,
//  copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
//  Software, and to permit persons to whom the Software is furnished to do so,
//  subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
//  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
//  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
//  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
//  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#ifndef VECTOR3D_H
#define VECTOR3D_H

#include <math.h>

#define DEGREE_TO_RAD       ((float)M_PI / 180.0f)
#define RAD_TO_DEGREE       (180.0f / (float)M_PI)

#define TWO_PI              (2.0f * (float)M_PI)

#define VEC3_X      0
```

```
#define VEC3_Y       1
#define VEC3_Z       2

typedef float vector3d_t[3];

void vector3DotProduct(vector3d_t a, vector3d_t b, float *d);
void vector3CrossProduct(vector3d_t a, vector3d_t b, vector3d_t d);

#endif
```

code/hydroAHRS_mkII/quaternion.c

```
////////////////////////////////////////////////////////////////////////////
//
//  This file is part of linux-mpu9150
//
//  Copyright (c) 2013 Pansenti, LLC
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy of
//  this software and associated documentation files (the "Software"), to deal in
//  the Software without restriction, including without limitation the rights to use,
//  copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
//  Software, and to permit persons to whom the Software is furnished to do so,
//  subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
//  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
//  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
//  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
//  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#include "quaternion.h"

void quaternionNorm(quaternion_t q, float *n)
{
    *n = sqrtf(q[QUAT_W] * q[QUAT_W] + q[QUAT_X] * q[QUAT_X] +
            q[QUAT_Y] * q[QUAT_Y] + q[QUAT_Z] * q[QUAT_Z]);
}

void quaternionNormalize(quaternion_t q)
{
    float length;

    quaternionNorm(q, &length);

    if (length == 0)
        return;

    q[QUAT_W] /= length;
    q[QUAT_X] /= length;
    q[QUAT_Y] /= length;
    q[QUAT_Z] /= length;
}

void quaternionToEuler(quaternion_t q, vector3d_t v)
{
    // fix roll near poles with this tolerance
    float pole = (float)M_PI / 2.0f - 0.05f;

    v[VEC3_Y] = asinf(2.0f * (q[QUAT_W] * q[QUAT_Y] - q[QUAT_X] * q[QUAT_Z]));

    if ((v[VEC3_Y] < pole) && (v[VEC3_Y] > -pole)) {
        v[VEC3_X] = atan2f(2.0f * (q[QUAT_Y] * q[QUAT_Z] + q[QUAT_W] * q[QUAT_X]),
                1.0f - 2.0f * (q[QUAT_X] * q[QUAT_X] + q[QUAT_Y] * q[QUAT_Y]));
    }

    v[VEC3_Z] = atan2f(2.0f * (q[QUAT_X] * q[QUAT_Y] + q[QUAT_W] * q[QUAT_Z]),
            1.0f - 2.0f * (q[QUAT_Y] * q[QUAT_Y] + q[QUAT_Z] * q[QUAT_Z]));
}

void eulerToQuaternion(vector3d_t v, quaternion_t q)
{
    float cosX2 = cosf(v[VEC3_X] / 2.0f);
    float sinX2 = sinf(v[VEC3_X] / 2.0f);
    float cosY2 = cosf(v[VEC3_Y] / 2.0f);
    float sinY2 = sinf(v[VEC3_Y] / 2.0f);
    float cosZ2 = cosf(v[VEC3_Z] / 2.0f);
    float sinZ2 = sinf(v[VEC3_Z] / 2.0f);

    q[QUAT_W] = cosX2 * cosY2 * cosZ2 + sinX2 * sinY2 * sinZ2;
    q[QUAT_X] = sinX2 * cosY2 * cosZ2 - cosX2 * sinY2 * sinZ2;
    q[QUAT_Y] = cosX2 * sinY2 * cosZ2 + sinX2 * cosY2 * sinZ2;
    q[QUAT_Z] = cosX2 * cosY2 * sinZ2 - sinX2 * sinY2 * cosZ2;

    quaternionNormalize(q);
}

void quaternionConjugate(quaternion_t s, quaternion_t d)
{
    d[QUAT_W] = s[QUAT_W];
    d[QUAT_X] = -s[QUAT_X];
    d[QUAT_Y] = -s[QUAT_Y];
    d[QUAT_Z] = -s[QUAT_Z];
}

void quaternionMultiply(quaternion_t qa, quaternion_t qb, quaternion_t qd)
{
    vector3d_t va;
    vector3d_t vb;
    float dotAB;
    vector3d_t crossAB;

    va[VEC3_X] = qa[QUAT_X];
    va[VEC3_Y] = qa[QUAT_Y];
    va[VEC3_Z] = qa[QUAT_Z];

    vb[VEC3_X] = qb[QUAT_X];
    vb[VEC3_Y] = qb[QUAT_Y];
    vb[VEC3_Z] = qb[QUAT_Z];

    vector3DotProduct(va, vb, &dotAB);
    vector3CrossProduct(va, vb, crossAB);

    qd[QUAT_W] = qa[QUAT_W] * qb[QUAT_W] - dotAB;
    qd[QUAT_X] = qa[QUAT_W] * vb[VEC3_X] + qb[QUAT_W] * va[VEC3_X] + crossAB[VEC3_X];
    qd[QUAT_Y] = qa[QUAT_W] * vb[VEC3_Y] + qb[QUAT_W] * va[VEC3_Y] + crossAB[VEC3_Y];
    qd[QUAT_Z] = qa[QUAT_W] * vb[VEC3_Z] + qb[QUAT_W] * va[VEC3_Z] + crossAB[VEC3_Z];
}
```

code/hydroAHRS_mkII/quaternion.h

```
////////////////////////////////////////////////////////////////////////////
//
//  This file is part of linux-mpu9150
//
//  Copyright (c) 2013 Pansenti, LLC
//
//  Permission is hereby granted, free of charge, to any person obtaining a copy of
//  this software and associated documentation files (the "Software"), to deal in
//  the Software without restriction, including without limitation the rights to use,
//  copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the
//  Software, and to permit persons to whom the Software is furnished to do so,
//  subject to the following conditions:
//
//  The above copyright notice and this permission notice shall be included in all
//  copies or substantial portions of the Software.
//
//  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
//  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
//  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
//   HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
//   OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
//   SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

#ifndef MPUQUATERNION_H
#define MPUQUATERNION_H

#include "vector3d.h"

#define QUAT_W      0
#define QUAT_X      1
#define QUAT_Y      2
#define QUAT_Z      3

typedef float quaternion_t[4];

void quaternionNormalize(quaternion_t q);
void quaternionToEuler(quaternion_t q, vector3d_t v);
void eulerToQuaternion(vector3d_t v, quaternion_t q);
void quaternionConjugate(quaternion_t s, quaternion_t d);
void quaternionMultiply(quaternion_t qa, quaternion_t qb, quaternion_t qd);


#endif /* MPUQUATERNION_H */
```

code/hydroAHRS_mkII/Descriptors.c

```
#include "Descriptors.h"

/*  Device descriptor structure. This descriptor, located in FLASH memory, describes
      the overall
 *  device characteristics, including the supported USB version, control endpoint size
      and the
 *  number of device configurations. The descriptor is read out by the USB host when
      the enumeration
 *  process begins.
 */
const USB_Descriptor_Device_t PROGMEM DeviceDescriptor = {
     .Header                 = {.Size = sizeof(USB_Descriptor_Device_t), .Type =
         DTYPE_Device},

     .USBSpecification       = VERSION_BCD(01.10),
     .Class                  = CDC_CSCP_CDCClass,
     .SubClass               = CDC_CSCP_NoSpecificSubclass,
     .Protocol               = CDC_CSCP_NoSpecificProtocol,

     .Endpoint0Size          = FIXED_CONTROL_ENDPOINT_SIZE,

     .VendorID               = 0x03EB,
     .ProductID              = 0x2044,
     .ReleaseNumber          = VERSION_BCD(00.01),

     .ManufacturerStrIndex   = STRING_ID_Manufacturer,
     .ProductStrIndex        = STRING_ID_Product,
     .SerialNumStrIndex      = USE_INTERNAL_SERIAL,

     .NumberOfConfigurations = FIXED_NUM_CONFIGURATIONS
};

/** Configuration descriptor structure. This descriptor, located in FLASH memory,
     describes the usage
 *  of the device in one of its supported configurations, including information about
      any device interfaces
 *  and endpoints. The descriptor is read out by the USB host during the enumeration
      process when selecting
 *  a configuration so that the host may correctly communicate with the USB device.
 */
const USB_Descriptor_Configuration_t PROGMEM ConfigurationDescriptor = {
     .Config = {
             .Header                 = {.Size = sizeof(
                 USB_Descriptor_Configuration_Header_t), .Type = DTYPE_Configuration},

			.TotalConfigurationSize = sizeof(USB_Descriptor_Configuration_t),
			.TotalInterfaces        = 2,

			.ConfigurationNumber    = 1,
			.ConfigurationStrIndex  = NO_DESCRIPTOR,

			.ConfigAttributes       = (USB_CONFIG_ATTR_RESERVED |
				USB_CONFIG_ATTR_SELFPOWERED),

			.MaxPowerConsumption    = USB_CONFIG_POWER_MA(100)
		},

	.CDC_CCI_Interface = {
			.Header                 = {.Size = sizeof(USB_Descriptor_Interface_t), .
				Type = DTYPE_Interface},

			.InterfaceNumber        = 0,
			.AlternateSetting       = 0,

			.TotalEndpoints         = 1,

			.Class                  = CDC_CSCP_CDCClass,
			.SubClass               = CDC_CSCP_ACMSubclass,
			.Protocol               = CDC_CSCP_ATCommandProtocol,

			.InterfaceStrIndex      = NO_DESCRIPTOR
		},

	.CDC_Functional_Header = {
			.Header                 = {.Size = sizeof(
				USB_CDC_Descriptor_FunctionalHeader_t), .Type = DTYPE_CSInterface},
			.Subtype                = CDC_DSUBTYPE_CSInterface_Header,

			.CDCSpecification       = VERSION_BCD(01.10),
		},

	.CDC_Functional_ACM = {
			.Header                 = {.Size = sizeof(
				USB_CDC_Descriptor_FunctionalACM_t), .Type = DTYPE_CSInterface},
			.Subtype                = CDC_DSUBTYPE_CSInterface_ACM,

			.Capabilities           = 0x06,
		},

	.CDC_Functional_Union = {
			.Header                 = {.Size = sizeof(
				USB_CDC_Descriptor_FunctionalUnion_t), .Type = DTYPE_CSInterface},
			.Subtype                = CDC_DSUBTYPE_CSInterface_Union,

			.MasterInterfaceNumber  = 0,
			.SlaveInterfaceNumber   = 1,
		},

	.CDC_NotificationEndpoint = {
			.Header                 = {.Size = sizeof(USB_Descriptor_Endpoint_t), .Type
				 = DTYPE_Endpoint},

			.EndpointAddress        = CDC_NOTIFICATION_EPADDR,
			.Attributes             = (EP_TYPE_INTERRUPT | ENDPOINT_ATTR_NO_SYNC |
				ENDPOINT_USAGE_DATA),
			.EndpointSize           = CDC_NOTIFICATION_EPSIZE,
			.PollingIntervalMS      = 0xFF
		},

	.CDC_DCI_Interface = {
			.Header                 = {.Size = sizeof(USB_Descriptor_Interface_t), .
				Type = DTYPE_Interface},

			.InterfaceNumber        = 1,
			.AlternateSetting       = 0,

			.TotalEndpoints         = 2,

            .Class                  = CDC_CSCP_CDCDataClass,
            .SubClass               = CDC_CSCP_NoDataSubclass,
            .Protocol               = CDC_CSCP_NoDataProtocol,

            .InterfaceStrIndex      = NO_DESCRIPTOR
        },

    .CDC_DataOutEndpoint = {
            .Header                 = {.Size = sizeof(USB_Descriptor_Endpoint_t), .Type
                 = DTYPE_Endpoint},

            .EndpointAddress        = CDC_RX_EPADDR,
            .Attributes             = (EP_TYPE_BULK | ENDPOINT_ATTR_NO_SYNC |
                ENDPOINT_USAGE_DATA),
            .EndpointSize           = CDC_TXRX_EPSIZE,
            .PollingIntervalMS      = 0x05
        },

    .CDC_DataInEndpoint = {
            .Header                 = {.Size = sizeof(USB_Descriptor_Endpoint_t), .Type
                 = DTYPE_Endpoint},

            .EndpointAddress        = CDC_TX_EPADDR,
            .Attributes             = (EP_TYPE_BULK | ENDPOINT_ATTR_NO_SYNC |
                ENDPOINT_USAGE_DATA),
            .EndpointSize           = CDC_TXRX_EPSIZE,
            .PollingIntervalMS      = 0x05
        }
};

/** Language descriptor structure. This descriptor, located in FLASH memory, is
    returned when the host requests
 *  the string descriptor with index 0 (the first index). It is actually an array of
     16-bit integers, which indicate
 *  via the language ID table available at USB.org what languages the device supports
     for its string descriptors.
 */
const USB_Descriptor_String_t PROGMEM LanguageString = {
    .Header                 = {.Size = USB_STRING_LEN(1), .Type = DTYPE_String},

    .UnicodeString          = {LANGUAGE_ID_ENG}
};

/** Manufacturer descriptor string. This is a Unicode string containing the
    manufacturer's details in human readable
 *  form, and is read out upon request by the host when the appropriate string ID is
     requested, listed in the Device
 *  Descriptor.
 */
const USB_Descriptor_String_t PROGMEM ManufacturerString = {
    .Header                 = {.Size = USB_STRING_LEN(6), .Type = DTYPE_String},

    .UnicodeString          = L"Epsiro"
};

/** Product descriptor string. This is a Unicode string containing the product's
    details in human readable form,
 *  and is read out upon request by the host when the appropriate string ID is
     requested, listed in the Device
 *  Descriptor.
 */
const USB_Descriptor_String_t PROGMEM ProductString = {
    .Header                 = {.Size = USB_STRING_LEN(25), .Type = DTYPE_String},

    .UnicodeString          = L"Inertial␣Measurement␣Unit"

};

/** This function is called by the library when in device mode, and must be overridden
    (see library "USB Descriptors"
 *  documentation) by the application code so that the address and size of a requested
     descriptor can be given
```

```c
 *  to the USB library. When the device receives a Get Descriptor request on the
    control endpoint, this function
 *  is called so that the descriptor details can be passed back and the appropriate
    descriptor sent back to the
 *  USB host.
 */
uint16_t CALLBACK_USB_GetDescriptor(const uint16_t wValue,
                                    const uint8_t wIndex,
                                    const void** const DescriptorAddress) {

    const uint8_t  DescriptorType   = (wValue >> 8);
    const uint8_t  DescriptorNumber = (wValue & 0xFF);

    const void* Address = NULL;
    uint16_t    Size    = NO_DESCRIPTOR;

    switch (DescriptorType) {
        case DTYPE_Device:
            Address = &DeviceDescriptor;
            Size    = sizeof(USB_Descriptor_Device_t);
            break;
        case DTYPE_Configuration:
            Address = &ConfigurationDescriptor;
            Size    = sizeof(USB_Descriptor_Configuration_t);
            break;
        case DTYPE_String:
            switch (DescriptorNumber) {
                case STRING_ID_Language:
                    Address = &LanguageString;
                    Size    = pgm_read_byte(&LanguageString.Header.Size);
                    break;
                case STRING_ID_Manufacturer:
                    Address = &ManufacturerString;
                    Size    = pgm_read_byte(&ManufacturerString.Header.Size);
                    break;
                case STRING_ID_Product:
                    Address = &ProductString;
                    Size    = pgm_read_byte(&ProductString.Header.Size);
                    break;
            }

            break;
    }

    *DescriptorAddress = Address;
    return Size;
}
```

code/hydroAHRS_mkII/Descriptors.h

```c
#ifndef _DESCRIPTORS_H_
#define _DESCRIPTORS_H_

/* Includes: */
#include <avr/pgmspace.h>

#include <LUFA/Drivers/USB/USB.h>

/* Macros: */
/** Endpoint address of the CDC device-to-host notification IN endpoint. */
#define CDC_NOTIFICATION_EPADDR        (ENDPOINT_DIR_IN  | 2)

/** Endpoint address of the CDC device-to-host data IN endpoint. */
#define CDC_TX_EPADDR                  (ENDPOINT_DIR_IN  | 3)

/** Endpoint address of the CDC host-to-device data OUT endpoint. */
#define CDC_RX_EPADDR                  (ENDPOINT_DIR_OUT | 4)

/** Size in bytes of the CDC device-to-host notification IN endpoint. */
#define CDC_NOTIFICATION_EPSIZE        8

/** Size in bytes of the CDC data IN and OUT endpoints. */
```

```
#define CDC_TXRX_EPSIZE                 16

/* Type Defines: */
/** Type define for the device configuration descriptor structure. This must be defined in the
 *  application code, as the configuration descriptor contains several sub-descriptors which
 *  vary between devices, and which describe the device's usage to the host.
 */
typedef struct {
    USB_Descriptor_Configuration_Header_t    Config;

    // CDC Control Interface
    USB_Descriptor_Interface_t               CDC_CCI_Interface;
    USB_CDC_Descriptor_FunctionalHeader_t    CDC_Functional_Header;
    USB_CDC_Descriptor_FunctionalACM_t       CDC_Functional_ACM;
    USB_CDC_Descriptor_FunctionalUnion_t     CDC_Functional_Union;
    USB_Descriptor_Endpoint_t                CDC_NotificationEndpoint;

    // CDC Data Interface
    USB_Descriptor_Interface_t               CDC_DCI_Interface;
    USB_Descriptor_Endpoint_t                CDC_DataOutEndpoint;
    USB_Descriptor_Endpoint_t                CDC_DataInEndpoint;
} USB_Descriptor_Configuration_t;

/** Enum for the device string descriptor IDs within the device. Each string descriptor should
 *  have a unique ID index associated with it, which can be used to refer to the string from
 *  other descriptors.
 */
enum StringDescriptors_t {
    STRING_ID_Language     = 0, /**< Supported Languages string descriptor ID (must be zero) */
    STRING_ID_Manufacturer = 1, /**< Manufacturer string ID */
    STRING_ID_Product      = 2, /**< Product string ID */
};

/* Function Prototypes: */
uint16_t CALLBACK_USB_GetDescriptor(const uint16_t wValue,
        const uint8_t wIndex,
        const void** const DescriptorAddress)
    ATTR_WARN_UNUSED_RESULT ATTR_NON_NULL_PTR_ARG(3);

#endif
```

code/hydroAHRS_mkII/LUFAConfig.h

```
#ifndef _LUFA_CONFIG_H_
#define _LUFA_CONFIG_H_

/* General USB Driver Related Tokens: */
  #define USE_STATIC_OPTIONS               (USB_DEVICE_OPT_FULLSPEED | USB_OPT_RC32MCLKSRC | USB_OPT_BUSEVENT_PRIHIGH)

/* USB Device Mode Driver Related Tokens: */
  #define USE_FLASH_DESCRIPTORS
  #define FIXED_CONTROL_ENDPOINT_SIZE      8
  #define FIXED_NUM_CONFIGURATIONS         1
  #define MAX_ENDPOINT_INDEX               4

#endif
```

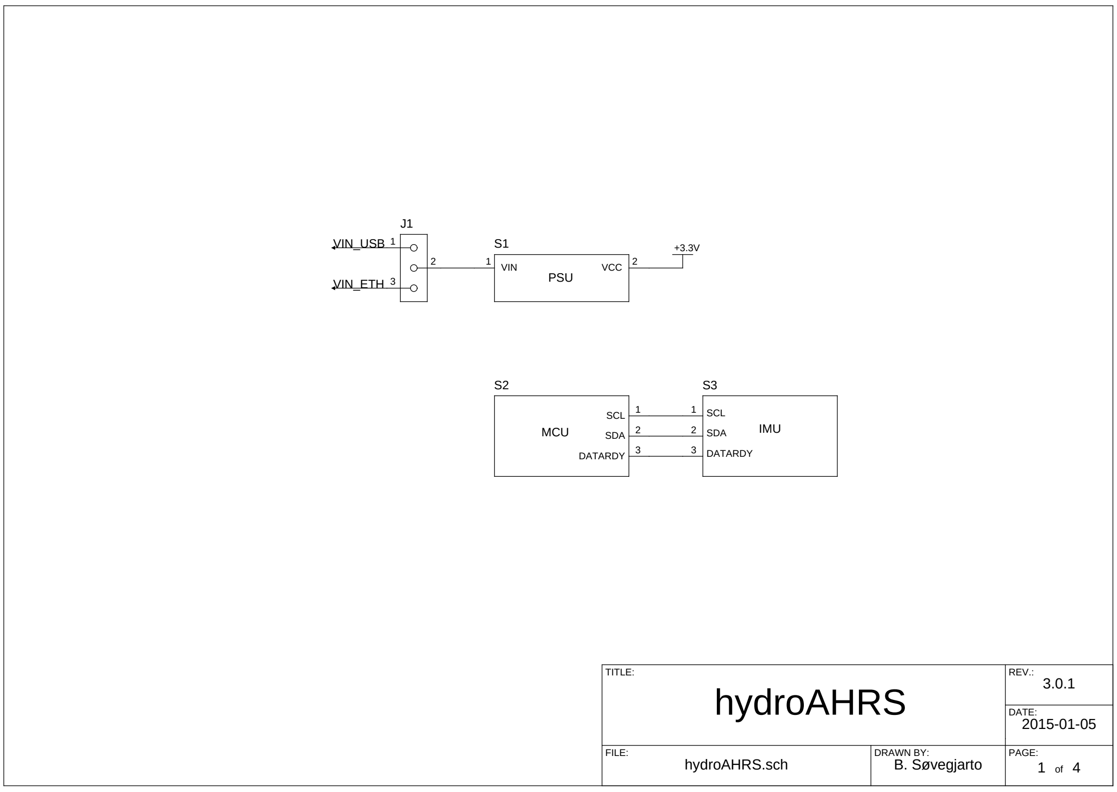
code/hydroAHRS_mkII/Makefile

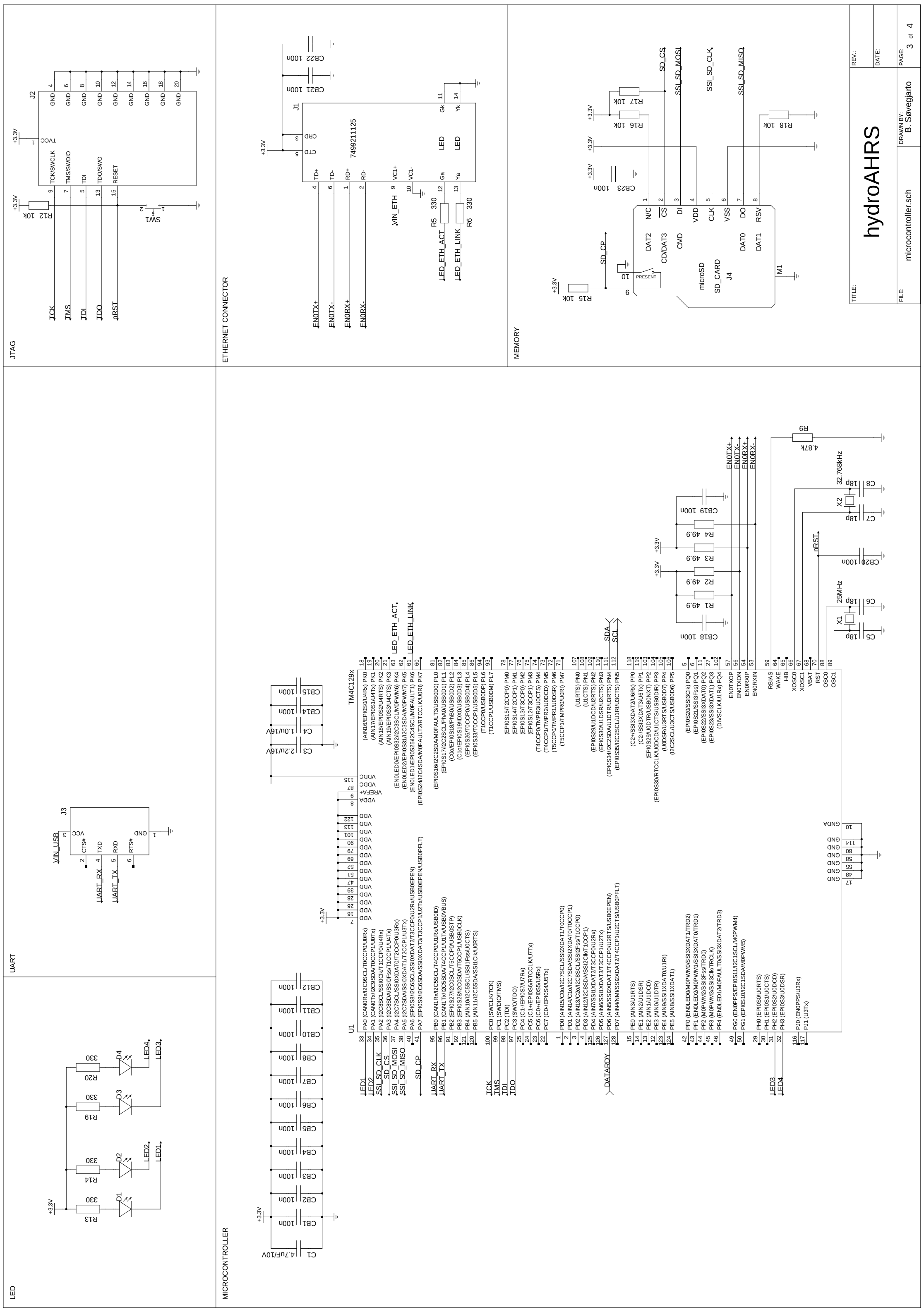
```
MCU         = atxmega128a4u
F_CPU       = 32000000
F_USB       = 48000000

AVRDUDE = avrdude -p x128a4u -c jtag2pdi -P usb
```

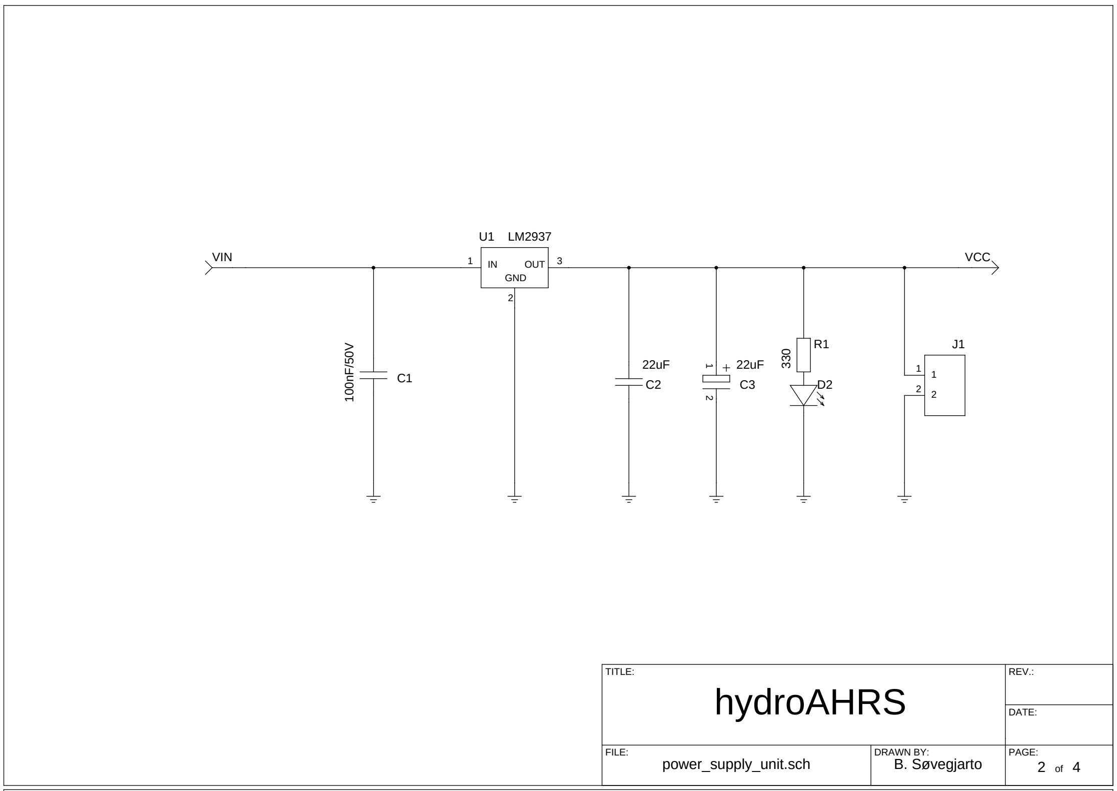
```
CC = avr-gcc
CC_FLAGS  = -DUSE_LUFA_CONFIG_HEADER
CC_FLAGS += -g -W -O2 -std=gnu99 -mmcu=$(MCU) -DF_CPU=$(F_CPU) -pedantic -mrelax
CC_FLAGS += -ffunction-sections
LD_FLAGS = -Wl,-gc-sections -Wl,-u,vfprintf -lprintf_flt -lm
DEFS = -DEMPL_TARGET_LINUX -DMPU9150 -DAK8975_SECONDARY -DTWIC

EMPLDIR = eMPL

SRC          = $(TARGET).c usb_talk.c Descriptors.c uart.c twi.c twim.c mpu9150.c quaternion.c vector3d.c inv_mpu.c inv_mpu_dmp_motion_driver.c nmea.c $(LUFA_SRC_USB) $(LUFA_SRC_USBCLASS)
TARGET       = main
ARCH         = XMEGA
OPTIMIZATION = s
LUFA_PATH    = ../LUFA-130901/LUFA

include $(LUFA_PATH)/Build/lufa_sources.mk
include $(LUFA_PATH)/Build/lufa_build.mk
include $(LUFA_PATH)/Build/lufa_hid.mk

flash:  all
	#$(AVRDUDE) -U flash:w:main.hex:i
	-echo "b" > /dev/ttyACM0
	sleep 3
	dfu-programmer atxmega128a4u erase
	dfu-programmer atxmega128a4u flash main.hex --debug 5
	! dfu-programmer atxmega128a4u start

inv_mpu_dmp_motion_driver.o : $(EMPLDIR)/inv_mpu_dmp_motion_driver.c
	$(CC) $(CC_FLAGS) $(DEFS) -I $(EMPLDIR) -c $(EMPLDIR)/inv_mpu_dmp_motion_driver.c

inv_mpu.o : $(EMPLDIR)/inv_mpu.c
	$(CC) $(CC_FLAGS) $(DEFS) -I $(EMPLDIR) -c $(EMPLDIR)/inv_mpu.c
```

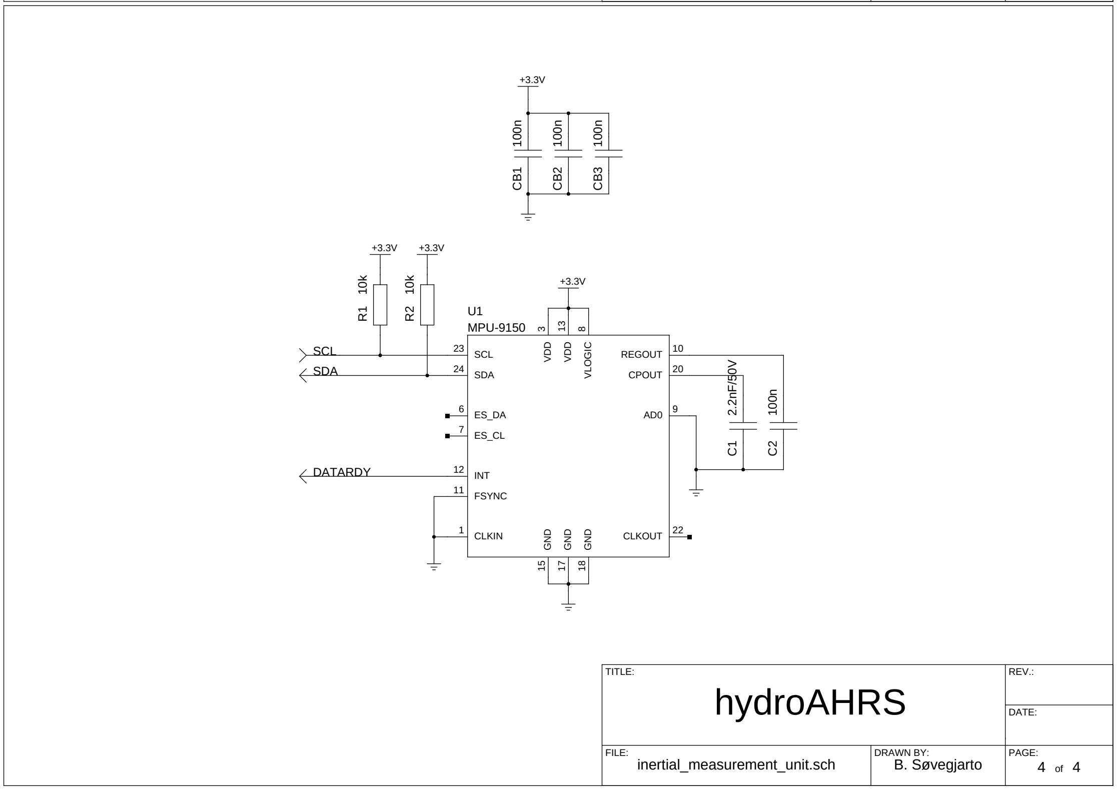
Appendix L

HydroAHRS mk.III schematics




JTAG
ETHERNET CONNECTOR
MEMORY
LED
UART
MICROCONTROLLER
TM4C129x
7499211125
microSD SD_CARD J4
hydroAHRS
B. Søvegjarto
microcontroller.sch
3 of 4


VIN
VCC
U1 LM2937
IN OUT GND
1 3 2
100nF/50V C1
22uF C2
22uF C3
330 R1
D2
J1
TITLE: hydroAHRS
REV.:
DATE:
FILE: power_supply_unit.sch
DRAWN BY: B. Søvegjarto
PAGE: 2 of 4


+3.3V
100n CB1
100n CB2
100n CB3
R1 10k
R2 10k
U1 MPU-9150
SCL
SDA
DATARDY
SCL SDA ES_DA ES_CL INT FSYNC CLKIN
VDD VDD VLOGIC
GND GND GND
REGOUT CPOUT AD0 CLKOUT
2.2nF/50V C1
100n C2
TITLE: hydroAHRS
REV.:
DATE:
FILE: inertial_measurement_unit.sch
DRAWN BY: B. Søvegjarto
PAGE: 4 of 4

Appendix M

MCU code for ASV autopilot

Code for TM4C129 ASV autopilot, as described in section 3.3.4. A block schematic of the algorithm running in the program is shown in figure 3.4.

This code depends on iot_mcu_lib and iot_tiva_template from Lindem Data Acquisition AS. iot_mcu_lib is available at `https://github.com/Lindem-Data-Acquisition-AS/iot_mcu_lib`. iot_tiva_template is available at `https://github.com/Lindem-Data-Acquisition-AS/iot_tiva_template`. The code is licensed under the MIT license (listing D.1) unless otherwise specified.

code/asv_ap_mcu/src/main.c

```c
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <stdio.h>
#include "inc/hw_gpio.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_types.h"
#include "inc/tm4c129xnczad.h"

#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "led_task.h"
#include "lwip_task.h"
#include "hello_world_task.h"
#include "asv_ap/uart.h"
#include "asv_ap/course_calc.h"

#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

uint32_t g_system_clock;

#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line) {
    send_debug_assert(pcFilename, ui32Line);
}
#endif

void
vApplicationStackOverflowHook(xTaskHandle *pxTask, signed char *pcTaskName) {

    ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, 0); // Red
    ROM_GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_2, 0); // Yellow

    while(1) {
    }

}

void
pin_init(void) {

    // Enable all the GPIO peripherals
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOR);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOS);

    // PF1/PK4/PK6 are used for Ethernet LEDs
    ROM_GPIOPinConfigure(GPIO_PK4_EN0LED0);
    ROM_GPIOPinConfigure(GPIO_PK6_EN0LED1);
    GPIOPinTypeEthernetLED(GPIO_PORTK_BASE, GPIO_PIN_4);
    GPIOPinTypeEthernetLED(GPIO_PORTK_BASE, GPIO_PIN_6);

#ifdef DEVKIT
    // PN5 is used for the user LED
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_5);
    ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_5, 0);
#else
    // PA0-1 is used for the user LED
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_1 | GPIO_PIN_0);
    ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1 | GPIO_PIN_0,  GPIO_PIN_0);
    // PH2-3 is used for the user LED
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_2);
    ROM_GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_3 | GPIO_PIN_2, GPIO_PIN_3 | GPIO_PIN_2)
        ;
#endif

}

int
main(void) {

    // Run from the PLL at 120 MHz.
    g_system_clock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                             SYSCTL_OSC_MAIN |
                                             SYSCTL_USE_PLL |
                                             SYSCTL_CFG_VCO_480),
                                            configCPU_CLOCK_HZ);

    MAP_FPULazyStackingEnable();
    MAP_FPUEnable();

    // UART init
    UART_init();

    // Enable processor interrupts.
    ROM_IntMasterEnable();

    // Initialize the device pinout appropriately for this board.
    pin_init();

    // Make sure the main oscillator is enabled because this is required by
    // the PHY.  The system must have a 25MHz crystal attached to the OSC
    // pins.  The SYSCTL_MOSC_HIGHFREQ parameter is used when the crystal
    // frequency is 10MHz or higher.
    SysCtlMOSCConfigSet(SYSCTL_MOSC_HIGHFREQ);

    printf("ASV␣autopilot␣started.\r\n");

    // Create the LED task.
    if (LEDTaskInit() != 0) {

        while (1) {
        }

    }

    // Create the lwIP tasks.
    if (lwIPTaskInit() != 0) {

        while (1) {
        }

    }

    // Create the hello world task.
    if (hello_world_init() != 0) {

        while (1) {
        }

    }

    // Create the ASV autopilot task.
    if (ASV_ap_init() != 0) {

        while (1) {
        }

    }

    // Start the scheduler. This should not return.
    vTaskStartScheduler();

    // In case the scheduler returns for some reason, loop forever.
    while (1) {
    }
}
```

code/asv_ap_mcu/src/main.h

```
extern uint32_t g_system_clock;
```

code/asv_ap_mcu/src/asv_ap/course_calc.c

```
#include <stdint.h>
#include <stdbool.h>
#include <inttypes.h>
#include <string.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "lwip/tcp.h"
#include "course_calc.h"
#include "driverlib/rom.h"
#include "uart.h"
#include "nmea.h"
#include "priorities.h"
#include "FreeRTOS.h"
#include "task.h"
```

```
#include "queue.h"
#include "semphr.h"
#include "udp_send.h"

#define STACKSIZE_ASV_AP_TASK         600

#define K_P      0.30
#define K_I      0.00
#define K_D      0.00

coord_t g_current_location;
coord_t g_first_waypoint = {0, 0};
coord_t g_next_waypoint = {0, 0};
float g_current_heading;
float g_wanted_heading;
uint32_t g_current_time;
uint32_t g_current_date;
uint8_t g_magnetometer_accuracy;
uint8_t g_run_motors = false;
uint8_t g_nr_of_wp = 0;
float g_distance_to_wp;
int8_t g_port_motor_thrust = 0;
int8_t g_stbd_motor_thrust = 0;
uint8_t g_send_status_msg = false;
uint8_t g_gps_ready = false;
uint8_t g_compass_ready = false;

static char message[4][UART_BUFFER_SIZE];


float
get_heading(coord_t initial_coordinate, coord_t final_coordinate) {

    float lat1 = initial_coordinate.latitude * DEG2RAD;
    float lat2 = final_coordinate.latitude * DEG2RAD;

    float delta_lon = (final_coordinate.longitude - initial_coordinate.longitude) *
        DEG2RAD;

    float y = sin(delta_lon) * cos(lat2);
    float x = cos(lat1)*sin(lat2) - sin(lat1)*cos(lat2)*cos(delta_lon);

    float heading = atan2(y, x);

    //          (heading*RAD2DEG + 360.0)% 360.0;
    return fmod((heading*RAD2DEG + 360.0), 360.0);
}

float
get_distance(coord_t initial_coordinate, coord_t final_coordinate) {

    uint32_t R = 6373;

    float lat1 = initial_coordinate.latitude * DEG2RAD;
    float lat2 = final_coordinate.latitude * DEG2RAD;

    float delta_lon = (final_coordinate.longitude - initial_coordinate.longitude) *
        DEG2RAD;
    float delta_lat = (final_coordinate.latitude - initial_coordinate.latitude) *
        DEG2RAD;

    float a = pow(sin(delta_lat/2), 2) + cos(lat1)*cos(lat2)*pow(sin(delta_lon/2), 2);
    float c = 2 * atan2(sqrt(a), sqrt(1 - a));
    float distance = R * c;

    return 1000*distance;
}

void
drive_motors(int8_t port_motor_int, int8_t stbd_motor_int) {

    /* Get the direction */
    uint8_t port_motor_direction = (port_motor_int > 0) ? 1 : 0;
    uint8_t stbd_motor_direction = (stbd_motor_int > 0) ? 1 : 0;

    /* Get the speed */
    uint8_t port_motor_speed = abs(port_motor_int);
    uint8_t stbd_motor_speed = abs(stbd_motor_int);

    /* Build the steering commands */
    uint8_t port_motor_cmd = (PORT_MOTOR << 7) | (port_motor_direction << 6) | (
        port_motor_speed & 0x3f);
    uint8_t stbd_motor_cmd = (STBD_MOTOR << 7) | (stbd_motor_direction << 6) | (
        stbd_motor_speed & 0x3f);

    /* Send the steering commands to the MC */
    ROM_UARTCharPut(MC_MODULE, port_motor_cmd);
    ROM_UARTCharPut(MC_MODULE, stbd_motor_cmd);
}

void
pid_init(int16_t p_factor, int16_t i_factor, int16_t d_factor, pid_data_t *pid) {

  // Start values for PID controller
  pid->sum_error = 0;
  pid->last_process_value = 0;

  // Tuning constants for PID loop
  pid->P_factor = p_factor;
  pid->I_factor = i_factor;
  pid->D_factor = d_factor;

  // Limits to avoid overflow
  pid->max_error = MAX_INT / (pid->P_factor + 1);
  pid->max_sum_error = MAX_I_TERM / (pid->I_factor + 1);
}

int16_t
modulo(int16_t a, int16_t n) {
   return a - floor(a/n) * n;
}

int16_t
angle_difference(int16_t a1, int16_t a2) {
    return modulo(((a1 - a2) + 180), 360) - 180;
}

int16_t
pid_controller(int16_t set_point, int16_t process_value, pid_data_t *pid) {

    int16_t error, p_term, d_term;
    int32_t i_term, ret, temp;

    error = angle_difference(set_point, process_value);

    // Calculate p_term and limit error overflow
    if (error > pid->max_error) {
      p_term = MAX_INT;

    } else if (error < -pid->max_error) {
      p_term = -MAX_INT;

    } else{
      p_term = pid->P_factor * error;
    }

    // Calculate i_term and limit integral runaway
    temp = pid->sum_error + error;

    if(temp > pid->max_sum_error) {
      i_term = MAX_I_TERM;
      pid->sum_error = pid->max_sum_error;

    } else if (temp < -pid->max_sum_error) {
      i_term = -MAX_I_TERM;
      pid->sum_error = -pid->max_sum_error;

    } else {
      pid->sum_error = temp;
      i_term = pid->I_factor * pid->sum_error;
    }

    // Calculate d_term
    d_term = pid->D_factor * (pid->last_process_value - process_value);

    pid->last_process_value = process_value;

    ret = (p_term + i_term + d_term) / SCALING_FACTOR;
    if (ret > MAX_INT) {
      ret = MAX_INT;
    } else if (ret < -MAX_INT) {
      ret = -MAX_INT;
    }

    return ( (int16_t) ret);
}

void
calculate_motor_thrust(pid_data_t *pid) {

    //int8_t constant_thrust = 0;
    int8_t constant_thrust = 32;

    int8_t port_motor_thrust;
    int8_t stbd_motor_thrust;
    int8_t diff_thrust;

    int8_t stbd_motor_sign;
    int8_t port_motor_sign;

    /* PID regulator */
    int16_t change = pid_controller(g_wanted_heading, g_current_heading, pid);

    /* Use different driving direction for each of the motors */
    stbd_motor_sign = (change > 0) ? -1 :  1;
    port_motor_sign = (change > 0) ?  1 : -1;

    /* Run both motors at the same speed if change is below threshold */
    if (abs(change) < 2) {
        diff_thrust = 0;

    /* Set lower bounds */
    } else if (abs(change) < 8) {
        diff_thrust = 8;

    /* Set upper bounds */
    } else if (abs(change) > 24) {
        diff_thrust = 24;

    } else {
        diff_thrust = abs(change);
    }

    stbd_motor_thrust = constant_thrust + diff_thrust*stbd_motor_sign;
    port_motor_thrust = constant_thrust + diff_thrust*port_motor_sign;

#if VERBOSE
    printf("Motor thrust (change, port, stbd): %i %i %i\n\r", change, port_motor_thrust
        , stbd_motor_thrust);
    fflush(stdout);
#endif

    g_compass_ready = true;
    if (g_run_motors && g_gps_ready && g_compass_ready) {

        g_stbd_motor_thrust = stbd_motor_thrust;
        g_port_motor_thrust = port_motor_thrust;

        drive_motors(port_motor_thrust, stbd_motor_thrust);
    } else {

        g_stbd_motor_thrust = 0;
        g_port_motor_thrust = 0;

        drive_motors(0, 0);
    }
}

void handle_mc_data() {

    get_uart_str(MC_DEVICE, message[MC_DEVICE]);

#ifdef DEBUG
    printf("MC␣␣:␣%s\r\n", message[MC_DEVICE]);
#endif
}

void handle_imu_data(pid_data_t *pid) {

    get_uart_str(IMU_DEVICE, message[IMU_DEVICE]);

#ifdef DEBUG
    printf("IMU␣:␣%s\r\n", message[IMU_DEVICE]);
#endif

    /* Check that the sentence is not corrupt */
    //if(nmea_checksum(message[GNSS_DEVICE]) == 0) {
        nmea_parse_PASHR(message[IMU_DEVICE]);

        if (g_magnetometer_accuracy == 3) {
            g_compass_ready = true;
        } else {
            g_compass_ready = false;
        }

#if VERBOSE
        printf("Course␣(wanted,␣current,␣error):␣%+07.2f␣%+07.2f␣%+07.2f\n\r",
               g_wanted_heading, g_current_heading, (g_wanted_heading -
                   g_current_heading));
        fflush(stdout);
#endif

        calculate_motor_thrust(pid);
        g_send_status_msg = true;
    //}
}

void handle_gnss_data() {
    get_uart_str(GNSS_DEVICE, message[GNSS_DEVICE]);

#ifdef DEBUG
    printf("GNSS:␣%s\r\n", message[GNSS_DEVICE]);
#endif

    /* Get the wanted message type */
    if (!strncmp(message[GNSS_DEVICE], "$GPRMC", 6)) {

        /* Check that the sentence is not corrupt */
        if(nmea_checksum(message[GNSS_DEVICE]) == 0) {

            /* And if it good, parse it and get the current location */
            nmea_parse_GPRMC(message[GNSS_DEVICE]);

            if (g_current_location.latitude == 0 && g_current_location.longitude == 0)
                {
                g_gps_ready = false;

            } else {

                g_gps_ready = true;

                g_distance_to_wp = get_distance(g_current_location, g_first_waypoint);

                if (g_distance_to_wp < 2) {

                    if (g_next_waypoint.latitude != 0 && g_next_waypoint.longitude !=
                        0) {
                        g_first_waypoint.latitude = g_next_waypoint.latitude;
                        g_first_waypoint.longitude = g_next_waypoint.longitude;
                        g_next_waypoint.latitude = 0;
                        g_next_waypoint.longitude = 0;

                    } else {
                        g_first_waypoint.latitude = 0;
                        g_first_waypoint.longitude = 0;

                        /* Stop motors if we do not have a waypoint to go to */
                        g_run_motors = false;
                    }
                }

                g_nr_of_wp = 0;
                if (g_first_waypoint.latitude != 0 && g_first_waypoint.longitude != 0)
                    g_nr_of_wp++;
                if (g_next_waypoint.latitude  != 0 && g_next_waypoint.longitude  != 0)
                    g_nr_of_wp++;

                /* We can now calculate our new course based on our current
                 * location and the first waypoint */
                g_wanted_heading = get_heading(g_current_location, g_first_waypoint);
            }
        }
    }
}

void
parse_command(struct tcp_pcb *tpcb, struct pbuf *p) {

    char incomming_command[512];
    char response[128];
    uint32_t index = 0;
    double latitude;
    double longitude;

    send_udp_bin((uint8_t *) p->payload, p->len);

    memcpy(&incomming_command, p->payload, p->len);
    printf("%s\r\n", incomming_command);

    /* Check message type*/
    if (!strncmp(&incomming_command[index], "HLT", 3)) {
        g_run_motors = false;

    } else if (!strncmp(&incomming_command[index], "RUN", 3)) {
        g_run_motors = true;

    } else if (!strncmp(&incomming_command[index], "DEL", 3)) {
        g_first_waypoint.latitude = 0;
        g_first_waypoint.longitude = 0;
        g_next_waypoint.latitude = 0;
        g_next_waypoint.longitude = 0;
        g_nr_of_wp = 0;

    } else if (!strncmp(&incomming_command[index], "NWP", 3)) {

        /* Skip to message */
        while (incomming_command[index++] != ',');

        // Get latitude [+-mmmm.mmmmm]
        latitude = atof(&incomming_command[index]);

        // Skip the latitude (already parsed)
        while (incomming_command[index++] != ',');

        // Get longitude [+-mmmm.mmmmm]
        longitude = atof(&incomming_command[index]);

        // Skip the longitude (already parsed)
        while (incomming_command[index++] != ',');

        // Acknowledge new WP
        sprintf(response, "ACKRWP,%.6f,%.6f", latitude, longitude);
        send_udp(response);
        // FIXME use tcp for ack. tcp_write(tpcb, &response, strlen(response), 1);

        if (g_first_waypoint.latitude == 0 && g_first_waypoint.longitude == 0) {
            g_first_waypoint.latitude = latitude;
            g_first_waypoint.longitude = longitude;

        } else {
            g_next_waypoint.latitude = latitude;
            g_next_waypoint.longitude = longitude;
        }

        g_nr_of_wp = 0;
        if (g_first_waypoint.latitude != 0 && g_first_waypoint.longitude != 0)
            g_nr_of_wp++;
        if (g_next_waypoint.latitude  != 0 && g_next_waypoint.longitude  != 0)
            g_nr_of_wp++;

        g_distance_to_wp = get_distance(g_current_location, g_first_waypoint);
    }
}

static void
ASV_ap_task(void *pvParameters) {

    char status[512];
    uint8_t gps_ok = 0;

    // Loop forever
    while (1) {

        vTaskDelay(100 / portTICK_RATE_MS);

        if (cmd_counter[MC_DEVICE] > 0) {
            handle_mc_data();
        }

        if (cmd_counter[IMU_DEVICE] > 0) {
            handle_imu_data(pvParameters);
        }

        if (cmd_counter[GNSS_DEVICE] > 0) {
            handle_gnss_data();
        }

        if (g_current_location.latitude != 0 && g_current_location.longitude != 0) {
            gps_ok = 1;
        } else {
            gps_ok = 0;
        }

        if (g_send_status_msg) {

            sprintf(status, "STATUS,%06lu,%06lu,%.6f,%.6f,%i,%.2f,%.2f,%.1f,%i,%d,%d
                ,%.6f,%.6f,%.6f,%.6f,%i,%i",
                (unsigned long) g_current_date,
                (unsigned long) g_current_time,
                g_current_location.latitude,
                g_current_location.longitude,
                gps_ok,
                g_current_heading,
                g_wanted_heading,
                g_distance_to_wp,
                g_nr_of_wp,
                g_magnetometer_accuracy,
                g_run_motors,
                g_first_waypoint.latitude,
                g_first_waypoint.longitude,
                g_next_waypoint.latitude,
                g_next_waypoint.longitude,
                g_port_motor_thrust,
                g_stbd_motor_thrust);

            //UART_send(LOG_MODULE, status);
            printf("%s\r\n", status);
            send_udp( status );

            g_send_status_msg = false;
        }
    }
}

uint32_t
ASV_ap_init(void) {

    // Parameters for regulator
    static pid_data_t pid_data;

    // PID init
    pid_init(K_P * SCALING_FACTOR, K_I * SCALING_FACTOR , K_D * SCALING_FACTOR , &
        pid_data);

    // Start AHRS
    //UART_send(IMU_MODULE, "s");

    // Create the ASV autopilot task.
    if (xTaskCreate(ASV_ap_task, (const portCHAR * const)"ASV_ap",
        STACKSIZE_ASV_AP_TASK, &pid_data, tskIDLE_PRIORITY + PRIORITY_ASV_AP_TASK, NULL
        ) != pdTRUE) {
        return(1);
    }

    // Success.
    return(0);
}
```

code/asv_ap_mcu/src/asv_ap/course_calc.h

```
#ifndef COURSE_CALC_H
#define COURSE_CALC_H

/** 239.255.66.83 */
//#define IPADDR_ASV_MULTICAST  ((u32_t)0xefff4253UL)
/** 192.168.0.19 */
//#define IPADDR_ASV_MULTICAST  ((u32_t)0xC0A80013UL)
#define IPADDR_ASV_MULTICAST  ((u32_t)0xffffffffUL)

#include <stdint.h>

#define SCALING_FACTOR  128

#define PORT_MOTOR 0x00
#define STBD_MOTOR 0x01

#define DEG2RAD (2*M_PI)/360.0
#define RAD2DEG 360.0/(2*M_PI)

typedef struct {
   double latitude;
   double longitude;
} coord_t;

extern coord_t g_current_location;
extern coord_t g_first_waypoint;
extern coord_t g_next_waypoint;

extern float g_current_heading;
extern float g_wanted_heading;

extern uint32_t g_current_time;
extern uint32_t g_current_date;

extern uint8_t g_magnetometer_accuracy;
extern uint8_t g_run_motors;

extern uint32_t hello_world_init(void);

/* PID Status
 * Setpoints and data used by the PID control algorithm
 */
typedef struct {

  // Last process value, used to find derivative of process value.
  int16_t last_process_value;

  // Summation of errors, used for integrate calculations
  int32_t sum_error;

  // The Proportional tuning constant, multiplied with SCALING_FACTOR
  int16_t P_factor;

  // The Integral tuning constant, multiplied with SCALING_FACTOR
  int16_t I_factor;

  // The Derivative tuning constant, multiplied with SCALING_FACTOR
  int16_t D_factor;

  // Maximum allowed error, avoid overflow
  int16_t max_error;

  // Maximum allowed sumerror, avoid overflow
  int32_t max_sum_error;

} pid_data_t;

/* Maximum values
 * Needed to avoid sign/overflow problems
 */
#define MAX_INT          INT16_MAX
#define MAX_LONG         INT32_MAX
#define MAX_I_TERM       (MAX_LONG / 2)

void pid_init(int16_t p_factor, int16_t i_factor, int16_t d_factor, pid_data_t *pid);
void handle_mc_data();
void handle_imu_data(pid_data_t *pid);
void handle_gnss_data();
void parse_command(struct tcp_pcb *tpcb, struct pbuf *p);
uint32_t ASV_ap_init(void);

#endif
```

code/asv_ap_mcu/src/asv_ap/nmea.c

```
#include "nmea.h"

#include <inttypes.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "course_calc.h"
#include "uart.h"

uint8_t
nmea_checksum(char nmea_sentence[]) {

    char checksum[3];

    uint8_t temp = 0;

    // 1 because we want to skip the $
    uint8_t i = 1;

    /* Calculate checksum */
    while(nmea_sentence[i] != '*')

          // XOR each character
          temp ^= nmea_sentence[i++];

    sprintf(checksum, "%02X\n", temp);

    /* Compare the two last characters in nmea_sentence with the checksum in (ascii) */
    if (nmea_sentence[++i] == checksum[0] &&
        nmea_sentence[++i] == checksum[1]) {
        return 0;
    } else {
        return 1;
    }
}

double
convert_from_degree_minutes_to_pure_degrees(double angle) {

    double degrees;
    double minutes_fraction;

    minutes_fraction = modf(angle/100, &degrees);
    angle = degrees + (minutes_fraction*100)/60;

    return angle;
}

void
nmea_parse_GPRMC(char nmea_sentence[]) {

    uint32_t index;
    float time;
    uint32_t date;
    double latitude;
    double longitude;

    index = 0;

    // Skip $GPRMC
    while (nmea_sentence[index++] != ',');

    // Get time
    time = atoi(&nmea_sentence[index]);

    // Skip time (already parsed)
    while (nmea_sentence[index++] != ',');

    // Skip A
    while (nmea_sentence[index++] != ',');

    // Get latitude [ddmm.mmmmm]
    latitude = atof(&nmea_sentence[index]);
    //printf("current_location n: %s\r\n", nmea_sentence[index]);
    //printf("current_location n: %d\r\n", (int) latitude);

    // Convert to pure degrees [dd.dddd] format
    latitude = convert_from_degree_minutes_to_pure_degrees(latitude);

    // Skip the latitude (already parsed)
    while (nmea_sentence[index++] != ',');

    // Correct latitude for N/S
    if (nmea_sentence[index] == 'S') {
        latitude = -latitude;
    }

    // Skip "N/S" (already parsed)
    while (nmea_sentence[index++] != ',');

    // Get longitude [ddmm.mmmmm]
    longitude = atof(&nmea_sentence[index]);

    // Convert to pure degrees [dd.dddd] format
    longitude = convert_from_degree_minutes_to_pure_degrees(longitude);

    // Skip the longitude (already parsed)
    while (nmea_sentence[index++] != ',');

    // Correct latitute for E/W
    if (nmea_sentence[index] == 'W') {
        longitude = -longitude;
    }

    // Skip "E/W" (already parsed)
    while (nmea_sentence[index++] != ',');

    // Skip speed
    while (nmea_sentence[index++] != ',');

    // Skip course
    while (nmea_sentence[index++] != ',');

    // Get date
    date = atoi(&nmea_sentence[index]);

    // Skip date (already parsed)
    while (nmea_sentence[index++] != ',');

    // Update the global variables
    g_current_location.latitude = latitude;
    g_current_location.longitude = longitude;
    g_current_time = (uint32_t) time;
    g_current_date = date;
}

void
nmea_parse_PASHR(char nmea_sentence[]) {
    //printf("current_location: %.6f,%.6f", latitude, longitude);

    float tilt;
    float roll;

    uint32_t index = 0;

    // Skip $PASHR
    while (nmea_sentence[index++] != ',');

    // Skip time
    while (nmea_sentence[index++] != ',');

    // Get heading
    g_current_heading = atof(&nmea_sentence[index]);
    while (nmea_sentence[index++] != ',');

    // Skip M
    while (nmea_sentence[index++] != ',');

    // Get tilt
    tilt = atof(&nmea_sentence[index]);
    while (nmea_sentence[index++] != ',');

    // Get roll
    tilt = atof(&nmea_sentence[index]);
    while (nmea_sentence[index++] != ',');

    // Skip empty
    while (nmea_sentence[index++] != ',');
    while (nmea_sentence[index++] != ',');
    while (nmea_sentence[index++] != ',');
```

```c
    while (nmea_sentence[index++] != ',');
    while (nmea_sentence[index++] != ',');

    // Get magnetometer accuracy
    g_magnetometer_accuracy = atoi(&nmea_sentence[index]);
}
```

code/asv_ap_mcu/src/asv_ap/nmea.h

```c
#ifndef NMEA_H
#define NMEA_H

#include <stdint.h>

uint8_t nmea_checksum(char nmea_sentence[]);
void nmea_parse_GPRMC(char nmea_sentence[]);
void nmea_parse_PASHR(char nmea_sentence[]);

#endif
```

code/asv_ap_mcu/src/asv_ap/uart.c

```c
#include "uart.h"

#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include <string.h>

#include "main.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/uart.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

/*
 * GNSS Tx: PD4 U2Rx 9600
 * GNSS Rx: PD5 U2Tx
 * AHRS Tx: PK0 U4Rx 115200
 * AHRS Rx: PK1 U4Tx
 * DBG  Tx: PC6 U5Rx 115200
 * DBG  Rx: PC7 U5Tx
 * MC   Tx: PC4 U7Rx 9600
 * MC   Rx: PC5 U7Tx
 *
 */

volatile char uart_buffer[4][UART_BUFFER_SIZE];
volatile uint8_t cmd_counter[4]  = {0, 0, 0, 0};
volatile uint8_t write_index[4]  = {0, 0, 0, 0};
volatile uint8_t read_index[4]   = {0, 0, 0, 0};
volatile uint8_t byte_counter[4] = {0, 0, 0, 0};

void
UART_init(void) {

    // Enable UART2, UART4, UART5, UART7 and GPIO peripherals
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART2);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART4);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART7);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

    // Set pin D4 and D5 as UART2
    ROM_GPIOPinConfigure(GPIO_PD4_U2RX);
    ROM_GPIOPinConfigure(GPIO_PD5_U2TX);
    ROM_GPIOPinTypeUART(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);

    // Set pin K0 and K1 as UART4
    ROM_GPIOPinConfigure(GPIO_PK0_U4RX);
    ROM_GPIOPinConfigure(GPIO_PK1_U4TX);
    ROM_GPIOPinTypeUART(GPIO_PORTK_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // Set pin C6 and C7 as UART5
    ROM_GPIOPinConfigure(GPIO_PC6_U5RX);
    ROM_GPIOPinConfigure(GPIO_PC7_U5TX);
    ROM_GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_6 | GPIO_PIN_7);

    // Set pin C4 and C5 as UART7
    ROM_GPIOPinConfigure(GPIO_PC4_U7RX);
    ROM_GPIOPinConfigure(GPIO_PC5_U7TX);
    ROM_GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5);

    // Configure UART2 as 9600 baud 8N1
    ROM_UARTConfigSetExpClk(UART2_BASE, g_system_clock, 9600,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));

    // Configure UART4 as 115200 baud 8N1
    ROM_UARTConfigSetExpClk(UART4_BASE, g_system_clock, 115200,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));

    // Configure UART5 as 115200 baud 8N1
    ROM_UARTConfigSetExpClk(UART5_BASE, g_system_clock, 115200,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));

    // Configure UART7 as 9600 baud 8N1
    ROM_UARTConfigSetExpClk(UART7_BASE, g_system_clock, 9600,
                            (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                             UART_CONFIG_PAR_NONE));


    // Enable UART interrupts
    ROM_IntEnable(INT_UART2);
    ROM_IntEnable(INT_UART4);
    ROM_IntEnable(INT_UART5);
    ROM_IntEnable(INT_UART7);
    ROM_UARTIntEnable(UART2_BASE, UART_INT_RX | UART_INT_RT);
    ROM_UARTIntEnable(UART4_BASE, UART_INT_RX | UART_INT_RT);
    ROM_UARTIntEnable(UART5_BASE, UART_INT_RX | UART_INT_RT);
    ROM_UARTIntEnable(UART7_BASE, UART_INT_RX | UART_INT_RT);
}

void
UART_send_nn(uint32_t module, char *string) {

    // Loop until string is over
    while (*string != '\0') {

        // Write single character
        ROM_UARTCharPut(module, *string++);
    }
}

void
UART_send(uint32_t module, char *string) {

    UART_send_nn(module, string);

    ROM_UARTCharPut(module, '\r');
    ROM_UARTCharPut(module, '\n');
}

char get_uart_char(uint8_t device) {

    char data;

    /* Wait for data from device */
    while (byte_counter[device] == 0);

    /* Disable interrupt because we want to access the uart buffer */
    ROM_IntMasterDisable();

    /* Copy a byte from the uart buffer to a local variable */
    data = uart_buffer[device][read_index[device]++];

    /* Reset the read counter if we have reached the end of the buffer,
     * the message contiues from the start of the memory */
    if (read_index[device] == UART_BUFFER_SIZE) {
        read_index[device] = 0;
    }

    /* We have finished getting one byte, so there is one less to do */
    byte_counter[device]--;

    /* We can enable interrupt again */
    ROM_IntMasterEnable();

    /* And return the data */
    return data;
}

void get_uart_str(uint8_t device, char *message_buffer) {
    uint8_t i = 0;

    for (i = 0; ( (message_buffer[i] = get_uart_char(device)) != '\n'); i++);

    /* Null terminate string */
    message_buffer[i] = 0;

    /* Decrement command counter (without beeing interrupted) */
    ROM_IntMasterDisable();
    cmd_counter[device]--;
    ROM_IntMasterEnable();
}

void
uart_rx_interrupt_handler(uint8_t device, uint32_t module) {

    uint32_t ui32Status;

    // Get the interrrupt status.
    ui32Status = ROM_UARTIntStatus(module, true);

    // Clear the asserted interrupts.
    ROM_UARTIntClear(module, ui32Status);

    // Loop while there are characters in the receive FIFO.
    while (ROM_UARTCharsAvail(module)) {

        char data = ROM_UARTCharGet(module);

        if (data != '\r') {

            uart_buffer[device][write_index[device]++] = data;

            if (data == '\n') {
                cmd_counter[device]++;
            }

            if (write_index[device] == UART_BUFFER_SIZE) {
                write_index[device] = 0;
            }

            byte_counter[device]++;

            /* Error handling */
            if (byte_counter[device] == UART_BUFFER_SIZE) {
                byte_counter[device] = 0;
                cmd_counter[device] = 0;
                }
            }
        }
}

void
UART5_int_handler(void) {

    uint32_t status;

    // Get the interrrupt status.
    status = ROM_UARTIntStatus(UART5_BASE, true);

    // Clear the asserted interrupts.
    ROM_UARTIntClear(UART5_BASE, status);

    // Loop while there are characters in the receive FIFO.
    while (ROM_UARTCharsAvail(UART5_BASE)) {
        ROM_UARTCharPut(UART5_BASE, ROM_UARTCharGet(UART5_BASE));
    }
}

void
UART2_int_handler(void) {
    uart_rx_interrupt_handler(GNSS_DEVICE, GNSS_MODULE);
}

void
UART4_int_handler(void) {
    uart_rx_interrupt_handler(IMU_DEVICE, IMU_MODULE);
}

void
UART7_int_handler(void) {
    uart_rx_interrupt_handler(MC_DEVICE, MC_MODULE);
}
```

code/asv_ap_mcu/src/asv_ap/uart.h

```
#include <stdint.h>
#include "inc/hw_memmap.h"

#define UART_BUFFER_SIZE 255

#define LOG_MODULE UART5_BASE
#define MC_MODULE UART7_BASE
#define IMU_MODULE UART4_BASE
#define GNSS_MODULE UART2_BASE
#define WLAN_MODULE UART5_BASE

//#define LOG_DEVICE 0x00
#define MC_DEVICE 0x00
#define IMU_DEVICE 0x01
#define GNSS_DEVICE 0x02
#define WLAN_DEVICE 0x03

void UART_init(void);
void UART_send(uint32_t module, char *string);
void UART_send_nn(uint32_t module, char *string);
void WLAN_send(char *string);
void get_uart_str(uint8_t device, char *message_buffer);
void UART2_int_handler(void);
void UART4_int_handler(void);
void UART5_int_handler(void);
void UART7_int_handler(void);

extern volatile uint8_t cmd_counter[];
```

code/asv_ap_mcu/src/tasks/led_task.c

```
/*
```

```c
 * Copyright (c) 2015 Lindem Data Acquisition AS. All rights reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you may not use
 * these files except in compliance with the License. You may obtain a copy of the
 * License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software distributed
 * under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
 * CONDITIONS OF ANY KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations under the License.
 *
 * Author:        Joakim Myrland
 * website:       www.LDA.as
 * email:         joakim.myrland@LDA.as
 * project:       https://github.com/Lindem-Data-Acquisition-AS/iot_tiva_template/
 *
 */

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/rom.h"
#include "config/lwiplib.h"
#include "led_task.h"
#include "priorities.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

// The stack size for the LED toggle task.
#define STACKSIZE_LEDTASK        128

// The amount of time to delay between toggles of the LED.
#define LED_DELAY_ON     750
#define LED_DELAY_OFF    250

// This task simply toggles the user LED at a 1 Hz rate.
static void
LEDTask(void *pvParameters) {

    portTickType ui32LastTime;
    uint32_t ui32Temp;

    // Get the current tick count.
    ui32LastTime = xTaskGetTickCount();

    // Loop forever.
    while (1) {

        ui32Temp = lwIPLocalIPAddrGet();

        /* No IP acquired */
        if (ui32Temp == IPADDR_NONE || ui32Temp == IPADDR_ANY) {

            // Turn off the user LED.
            #ifdef DEVKIT
                ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_5, GPIO_PIN_5);
            #else
                ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, GPIO_PIN_1);
            #endif

            // Wait for the required amount of time.
            vTaskDelayUntil(&ui32LastTime, LED_DELAY_OFF / portTICK_RATE_MS);

            // Turn on the user LED.
            #ifdef DEVKIT
                ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_5, 0);
            #else
```

```
                ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, 0);
            #endif

            // Wait for the required amount of time.
            vTaskDelayUntil(&ui32LastTime, LED_DELAY_OFF / portTICK_RATE_MS);

        /* Auto IP acquired */
        } else if ( (ui32Temp & 0xFFFF) == 0xFEA9 ) {

            // Turn off the user LED.
            #ifdef DEVKIT
                ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_5, GPIO_PIN_5);
            #else
                ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, GPIO_PIN_1);
            #endif

            // Wait for the required amount of time.
            vTaskDelayUntil(&ui32LastTime, LED_DELAY_ON / portTICK_RATE_MS);

            // Turn on the user LED.
            #ifdef DEVKIT
                ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_5, 0);
            #else
                ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, 0);
            #endif

            // Wait for the required amount of time.
            vTaskDelayUntil(&ui32LastTime, LED_DELAY_OFF / portTICK_RATE_MS);

        /* DHCP IP acquired */
        } else if ( (ui32Temp & 0xFFFF) == 0xA8C0 ) {

            // Turn off the user LED.
            #ifdef DEVKIT
                ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_5, GPIO_PIN_5);
            #else
                ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, GPIO_PIN_1);
            #endif

            // Wait for the required amount of time.
            vTaskDelayUntil(&ui32LastTime, LED_DELAY_OFF / portTICK_RATE_MS);

            // Turn on the user LED.
            #ifdef DEVKIT
                ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_5, 0);
            #else
                ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, 0);
            #endif

            // Wait for the required amount of time.
            vTaskDelayUntil(&ui32LastTime, LED_DELAY_ON / portTICK_RATE_MS);

        } else {
            vTaskDelayUntil(&ui32LastTime, 2000 / portTICK_RATE_MS);
        }

    }

}

// Initializes the LED task.
uint32_t
LEDTaskInit(void) {

    // Create the LED task.
    if (xTaskCreate(LEDTask, (const portCHAR * const)"LED", STACKSIZE_LEDTASK, NULL,
        tskIDLE_PRIORITY + PRIORITY_LED_TASK, NULL) != pdTRUE) {
        return(1);
    }

    // Success.
    return(0);
}
```

code/asv_ap_mcu/src/tasks/led_task.h

```c
/*
 * Copyright (c) 2015 Lindem Data Acquisition AS. All rights reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License"); you may not use
 * these files except in compliance with the License. You may obtain a copy of the
 * License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software distributed
 * under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
 * CONDITIONS OF ANY KIND, either express or implied. See the License for the
 * specific language governing permissions and limitations under the License.
 *
 * Author:        Joakim Myrland
 * website:       www.LDA.as
 * email:         joakim.myrland@LDA.as
 * project:       https://github.com/Lindem-Data-Acquisition-AS/iot_tiva_template/
 *
 */

#ifndef __LED_TASK_H__
#define __LED_TASK_H__

//*****************************************************************************
//
// Prototypes for the LED task.
//
//*****************************************************************************
extern uint32_t g_ui32LEDDelay;
extern uint32_t LEDTaskInit(void);

#endif // __LED_TASK_H__
```

code/asv_ap_mcu/src/tasks/lwip_task.c

```c
//*****************************************************************************
//
// lwip_task.c - Tasks to serve web pages over Ethernet using lwIP.
//
// Copyright (c) 2009-2014 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.0.12573 of the DK-TM4C129X Firmware Package.
//
//*****************************************************************************

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "driverlib/rom.h"
#include "lwiplib.h"
#include "httpserver_raw/httpd.h"
#include "httpserver_raw/fsdata.h"
#include "led_task.h"
#include "lwip_task.h"

```

```c
extern uint32_t g_system_clock;

//*****************************************************************************
//
// Sets up the additional lwIP raw API services provided by the application.
//
//*****************************************************************************
void
SetupServices(void *pvArg)
{
    //
    // Initialize the sample httpd server.
    //
    httpd_init();

    /* Initalize the tcp config server */
    config_server_init();

    //http_server_netconn_init();

}

//*****************************************************************************
//
// Initializes the lwIP tasks.
//
//*****************************************************************************
uint32_t
lwIPTaskInit(void)
{
    uint8_t pui8MAC[6];
    uint8_t hex_array[16] = {
        '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'
    };

    uint32_t ui32User0, ui32User1;
    ROM_FlashUserGet(&ui32User0, &ui32User1);

    if((ui32User0 == 0xffffffff) || (ui32User1 == 0xffffffff)) {

        // MAC address has not been programmed, use default.
        // ASV = 0x41 0x53 0x56
        pui8MAC[0] = 0x02;
        pui8MAC[1] = 0x41;
        pui8MAC[2] = 0x53;
        pui8MAC[3] = 0x56;
        pui8MAC[4] = 0x00;
        pui8MAC[5] = 0x00;

    } else {

        pui8MAC[0] = ((ui32User0 >>  0) & 0xff);
        pui8MAC[1] = ((ui32User0 >>  8) & 0xff);
        pui8MAC[2] = ((ui32User0 >> 16) & 0xff);
        pui8MAC[3] = ((ui32User1 >>  0) & 0xff);
        pui8MAC[4] = ((ui32User1 >>  8) & 0xff);
        pui8MAC[5] = ((ui32User1 >> 16) & 0xff);

    }

    //
    // Lower the priority of the Ethernet interrupt handler.  This is required
    // so that the interrupt handler can safely call the interrupt-safe
    // FreeRTOS functions (specifically to send messages to the queue).
    //
    ROM_IntPrioritySet(INT_EMAC0, 0xC0);

    //
    // Initialize lwIP.
    //
    //lwIPInit(g_system_clock, pui8MAC, aton("192.168.1.21"), aton("255.255.255.0"),
        aton("192.168.1.1"), IPADDR_USE_STATIC);
    lwIPInit(g_system_clock, pui8MAC, 0xC0A80115, 0xFFFFFF00, 0xC0A80101,
        IPADDR_USE_STATIC);
    //lwIPInit(g_system_clock, pui8MAC, 0xC0A80015, 0xFFFFFF00, 0xC0A80001,
        IPADDR_USE_STATIC);
    //lwIPInit(g_system_clock, pui8MAC, 0xC0A80015, 0xFFFFFF00, 0xC0A80014,
        IPADDR_USE_STATIC);

    //
    // Setup the remaining services inside the TCP/IP thread's context.
    //
    tcpip_callback(SetupServices, 0);

    //
    // Success.
    //
    return(0);
}
```

code/asv_ap_mcu/src/tasks/lwip_task.h

```
//*****************************************************************************
//
// lwip_task.h - Prototypes for the lwIP tasks.
//
// Copyright (c) 2009-2014 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.0.12573 of the DK-TM4C129X Firmware Package.
//
//*****************************************************************************

#ifndef __LWIP_TASK_H__
#define __LWIP_TASK_H__

//*****************************************************************************
//
// Prototypes.
//
//*****************************************************************************
extern uint32_t lwIPTaskInit(void);

#endif // __LWIP_TASK_H__
```

code/asv_ap_mcu/src/tasks/lwiplib.c

```
//*****************************************************************************
//
//
//
// Copyright (c) 2013-2014 Texas Instruments Incorporated.  All rights reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
```

```
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.0.12573 of the DK-TM4C129X Firmware Package.
//
//*****************************************************************************

#include <stdint.h>
#include <stdbool.h>

#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_emac.h"
#include "driverlib/debug.h"
#include "driverlib/emac.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"

#if RTOS_FREERTOS
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"
#endif

#include "lwiplib.h"
#include "lwip/tcpip.h"
#include "netif/tivaif.h"


//*****************************************************************************
//
// Ensure that ICMP checksum offloading is enabled; otherwise the TM4C129
// driver will not operate correctly.
//
//*****************************************************************************
#ifndef LWIP_OFFLOAD_ICMP_CHKSUM
#define LWIP_OFFLOAD_ICMP_CHKSUM 1
#endif


//*****************************************************************************
//
// The lwIP Library abstration layer provides for a host callback function to
// be called periodically in the lwIP context.  This is the timer interval, in
// ms, for this periodic callback.  If the timer interval is defined to 0 (the
// default value), then no periodic host callback is performed.
//
//*****************************************************************************
#ifndef HOST_TMR_INTERVAL
#define HOST_TMR_INTERVAL       0
#else
extern void lwIPHostTimerHandler(void);
#endif

//*****************************************************************************
//
// The link detect polling interval.
//
//*****************************************************************************
#define LINK_TMR_INTERVAL       10

//*****************************************************************************
//
// Set the PHY configuration to the default (internal) option if necessary.
//
//*****************************************************************************
```

```
#ifndef EMAC_PHY_CONFIG
#define EMAC_PHY_CONFIG            (EMAC_PHY_TYPE_INTERNAL |                        \
                                    EMAC_PHY_INT_MDIX_EN |                          \
                                    EMAC_PHY_AN_100B_T_FULL_DUPLEX)
#endif


// The lwIP network interface structure for the Tiva Ethernet MAC.
static struct netif g_sNetIF;
// The application's interrupt handler for hardware timer events from the MAC.
tHardwareTimerHandler g_pfnTimerHandler;
// The default IP address acquisition mode.
static uint32_t g_ui32IPMode = IPADDR_USE_STATIC;

// The most recently detected link state.
#if LWIP_AUTOIP || LWIP_DHCP
static bool g_bLinkActive = false;
#endif

// The IP address to be used.  This is used during the initialization of the
// stack and when the interface configuration is changed.
static uint32_t g_ui32IPAddr;
// The netmask to be used.  This is used during the initialization of the stack
// and when the interface configuration is changed.
static uint32_t g_ui32NetMask;
// The gateway address to be used.  This is used during the initialization of
// the stack and when the interface configuration is changed.
static uint32_t g_ui32GWAddr;

// The stack size for the interrupt task.
#if !NO_SYS
#define STACKSIZE_LWIPINTTASK   128

// The handle for the "queue" (semaphore) used to signal the interrupt task
// from the interrupt handler.
static xQueueHandle g_pInterrupt;

// This task handles reading packets from the Ethernet controller and supplying
// them to the TCP/IP thread.
static void
lwIPInterruptTask(void *pvArg)
{
    //
    // Loop forever.
    //
    while(1)
    {
        //
        // Wait until the semaphore has been signaled.
        //
        while(xQueueReceive(g_pInterrupt, &pvArg, portMAX_DELAY) != pdPASS)
        {
        }

        //
        // Processes any packets waiting to be sent or received.
        //
        tivaif_interrupt(&g_sNetIF, (uint32_t)pvArg);

        //
        // Re-enable the Ethernet interrupts.
        //
        MAP_EMACIntEnable(EMAC0_BASE, (EMAC_INT_RECEIVE | EMAC_INT_TRANSMIT |
                                       EMAC_INT_TX_STOPPED |
                                       EMAC_INT_RX_NO_BUFFER |
                                       EMAC_INT_RX_STOPPED | EMAC_INT_PHY));
    }
}
#endif

// This function performs a periodic check of the link status and responds
// appropriately if it has changed.
#if LWIP_AUTOIP || LWIP_DHCP
```

```
static void
lwIPLinkDetect(void)
{
    bool bHaveLink;
    struct ip_addr ip_addr;
    struct ip_addr net_mask;
    struct ip_addr gw_addr;

    //
    // See if there is an active link.
    //
    bHaveLink = MAP_EMACPHYRead(EMAC0_BASE, 0, EPHY_BMSR) & EPHY_BMSR_LINKSTAT;

    //
    // Return without doing anything else if the link state hasn't changed.
    //
    if(bHaveLink == g_bLinkActive)
    {
        return;
    }

    //
    // Save the new link state.
    //
    g_bLinkActive = bHaveLink;

    //
    // Clear any address information from the network interface.
    //
    ip_addr.addr = 0;
    net_mask.addr = 0;
    gw_addr.addr = 0;
    netif_set_addr(&g_sNetIF, &ip_addr, &net_mask, &gw_addr);

    //
    // See if there is a link now.
    //
    if(bHaveLink)
    {
        //
        // Start DHCP, if enabled.
        //
#if LWIP_DHCP
        if(g_ui32IPMode == IPADDR_USE_DHCP)
        {
            dhcp_start(&g_sNetIF);
        }
#endif

        //
        // Start AutoIP, if enabled and DHCP is not.
        //
#if LWIP_AUTOIP
        if(g_ui32IPMode == IPADDR_USE_AUTOIP)
        {
            autoip_start(&g_sNetIF);
        }
#endif
    }
    else
    {
        //
        // Stop DHCP, if enabled.
        //
#if LWIP_DHCP
        if(g_ui32IPMode == IPADDR_USE_DHCP)
        {
            dhcp_stop(&g_sNetIF);
        }
#endif

        //
        // Stop AutoIP, if enabled and DHCP is not.
        //
#if LWIP_AUTOIP
        if(g_ui32IPMode == IPADDR_USE_AUTOIP)
        {
            autoip_stop(&g_sNetIF);
        }
#endif
    }
}
#endif

// Handles the timeout for the host callback function timer when using a RTOS.
#if !NO_SYS && HOST_TMR_INTERVAL
static void
lwIPPrivateHostTimer(void *pvArg)
{
    //
    // Call the application-supplied host timer callback function.
    //
    lwIPHostTimerHandler();

    //
    // Re-schedule the host timer callback function timeout.
    //
    sys_timeout(HOST_TMR_INTERVAL, lwIPPrivateHostTimer, NULL);
}
#endif

// Handles the timeout for the link detect timer when using a RTOS.
#if !NO_SYS && (LWIP_AUTOIP || LWIP_DHCP)
static void
lwIPPrivateLinkTimer(void *pvArg)
{
    //
    // Perform the link detection.
    //
    lwIPLinkDetect();

    //
    // Re-schedule the link detect timer timeout.
    //
    sys_timeout(LINK_TMR_INTERVAL, lwIPPrivateLinkTimer, NULL);
}
#endif

// Completes the initialization of lwIP.  This is directly called when not
// using a RTOS and provided as a callback to the TCP/IP thread when using a
// RTOS.
static void
lwIPPrivateInit(void *pvArg)
{
    struct ip_addr ip_addr;
    struct ip_addr net_mask;
    struct ip_addr gw_addr;

#if !NO_SYS
#if RTOS_FREERTOS
    // If using a RTOS, create a queue (to be used as a semaphore) to signal
    // the Ethernet interrupt task from the Ethernet interrupt handler.
    g_pInterrupt = xQueueCreate(1, sizeof(void *));
    // If using a RTOS, create the Ethernet interrupt task.
    xTaskCreate(lwIPInterruptTask, (const portCHAR * const)"eth_isr",
                STACKSIZE_LWIPINTTASK, 0, tskIDLE_PRIORITY + 1,
                0);
#endif
#endif

    //
    // Setup the network address values.
    //
    if(g_ui32IPMode == IPADDR_USE_STATIC)
    {
        ip_addr.addr = htonl(g_ui32IPAddr);
        net_mask.addr = htonl(g_ui32NetMask);
        gw_addr.addr = htonl(g_ui32GWAddr);
    }
    else
    {
        ip_addr.addr = 0;
        net_mask.addr = 0;
        gw_addr.addr = 0;
    }

    //
    // Create, configure and add the Ethernet controller interface with
    // default settings.  ip_input should be used to send packets directly to
    // the stack when not using a RTOS and tcpip_input should be used to send
    // packets to the TCP/IP thread's queue when using a RTOS.
    //
    netif_add(&g_sNetIF, &ip_addr, &net_mask, &gw_addr, NULL, tivaif_init,
              tcpip_input);
    netif_set_default(&g_sNetIF);

    //
    // Bring the interface up.
    //
    netif_set_up(&g_sNetIF);

    //
    // Setup a timeout for the host timer callback function if using a RTOS.
    //
#if !NO_SYS && HOST_TMR_INTERVAL
    sys_timeout(HOST_TMR_INTERVAL, lwIPPrivateHostTimer, NULL);
#endif

    //
    // Setup a timeout for the link detect callback function if using a RTOS.
    //
#if !NO_SYS && (LWIP_AUTOIP || LWIP_DHCP)
    sys_timeout(LINK_TMR_INTERVAL, lwIPPrivateLinkTimer, NULL);
#endif
}


/*********************************************************************/
/*  IPIsNetIfUp                                                      */
/*                                                                   */
/*  Check if the interface and link is Up.                           */
/*                                                                   */
/*  In     : none                                                    */
/*  Out    : none                                                    */
/*  Return: 0 / 1                                                    */
/*********************************************************************/
int IPIsNetIfUp (void)
{
    int NetIfUp = 0;

    if ((netif_is_up(&g_sNetIF)) && (netif_is_link_up(&g_sNetIF)))
    {
        NetIfUp = 1;
    }

    return(NetIfUp);
} /* IPIsNetIfUp */

//*****************************************************************************
//
//! Initializes the lwIP TCP/IP stack.
//!
//! \param ui32SysClkHz is the current system clock rate in Hz.
//! \param pui8MAC is a pointer to a six byte array containing the MAC
//! address to be used for the interface.
//! \param ui32IPAddr is the IP address to be used (static).
//! \param ui32NetMask is the network mask to be used (static).
//! \param ui32GWAddr is the Gateway address to be used (static).
//! \param ui32IPMode is the IP Address Mode.  \b IPADDR_USE_STATIC will force
//! static IP addressing to be used, \b IPADDR_USE_DHCP will force DHCP with
//! fallback to Link Local (Auto IP), while \b IPADDR_USE_AUTOIP will force
//! Link Local only.
//!
//! This function performs initialization of the lwIP TCP/IP stack for the
//! Ethernet MAC, including DHCP and/or AutoIP, as configured.
//!
//! \return None.
//
//*****************************************************************************
void
lwIPInit(uint32_t ui32SysClkHz, const uint8_t *pui8MAC, uint32_t ui32IPAddr,
         uint32_t ui32NetMask, uint32_t ui32GWAddr, uint32_t ui32IPMode)
{
    //
    // Check the parameters.
    //
#if LWIP_DHCP && LWIP_AUTOIP
    ASSERT((ui32IPMode == IPADDR_USE_STATIC) ||
           (ui32IPMode == IPADDR_USE_DHCP) ||
           (ui32IPMode == IPADDR_USE_AUTOIP));
#elif LWIP_DHCP
    ASSERT((ui32IPMode == IPADDR_USE_STATIC) ||
           (ui32IPMode == IPADDR_USE_DHCP));
#elif LWIP_AUTOIP
    ASSERT((ui32IPMode == IPADDR_USE_STATIC) ||
           (ui32IPMode == IPADDR_USE_AUTOIP));
#else
    ASSERT(ui32IPMode == IPADDR_USE_STATIC);
#endif

    //
    // Enable the ethernet peripheral.
    //
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_EMAC0);
    MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_EMAC0);

    //
    // Enable the internal PHY if it's present and we're being
    // asked to use it.
    //
    if((EMAC_PHY_CONFIG & EMAC_PHY_TYPE_MASK) == EMAC_PHY_TYPE_INTERNAL)
    {
        //
        // We've been asked to configure for use with the internal
        // PHY.  Is it present?
        //
        if(MAP_SysCtlPeripheralPresent(SYSCTL_PERIPH_EPHY0))
        {
            //
            // Yes - enable and reset it.
            //
            MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_EPHY0);
            MAP_SysCtlPeripheralReset(SYSCTL_PERIPH_EPHY0);
        }
        else
        {
            //
            // Internal PHY is not present on this part so hang here.
            //
            while(1)
            {
            }
        }
    }

    //
    // Wait for the MAC to come out of reset.
    //
    while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_EMAC0))
    {
    }

    //
    // Configure for use with whichever PHY the user requires.
    //
    MAP_EMACPHYConfigSet(EMAC0_BASE, EMAC_PHY_CONFIG);

    //
    // Initialize the MAC and set the DMA mode.
    //
    MAP_EMACInit(EMAC0_BASE, ui32SysClkHz,
                 EMAC_BCONFIG_MIXED_BURST | EMAC_BCONFIG_PRIORITY_FIXED,
                 4, 4, 0);

    //
    // Set MAC configuration options.
    //
    MAP_EMACConfigSet(EMAC0_BASE, (EMAC_CONFIG_FULL_DUPLEX |
                                   EMAC_CONFIG_CHECKSUM_OFFLOAD |
                                   EMAC_CONFIG_7BYTE_PREAMBLE |
                                   EMAC_CONFIG_IF_GAP_96BITS |
                                   EMAC_CONFIG_USE_MACADDR0 |
                                   EMAC_CONFIG_SA_FROM_DESCRIPTOR |
                                   EMAC_CONFIG_BO_LIMIT_1024),
                      (EMAC_MODE_RX_STORE_FORWARD |
                       EMAC_MODE_TX_STORE_FORWARD |
                       EMAC_MODE_TX_THRESHOLD_64_BYTES |
                       EMAC_MODE_RX_THRESHOLD_64_BYTES), 0);

    //
    // Program the hardware with its MAC address (for filtering).
    //
    MAP_EMACAddrSet(EMAC0_BASE, 0, (uint8_t *)pui8MAC);

    //
    // Save the network configuration for later use by the private
    // initialization.
    //
    g_ui32IPMode = ui32IPMode;
    g_ui32IPAddr = ui32IPAddr;
    g_ui32NetMask = ui32NetMask;
    g_ui32GWAddr = ui32GWAddr;

    //
    // Initialize lwIP.  The remainder of initialization is done immediately if
    // not using a RTOS and it is deferred to the TCP/IP thread's context if
    // using a RTOS.
    //
    tcpip_init(lwIPPrivateInit, 0);
}

//*****************************************************************************
//
//! Registers an interrupt callback function to handle the IEEE-1588 timer.
//!
//! \param pfnTimerFunc points to a function which is called whenever the
//! Ethernet MAC reports an interrupt relating to the IEEE-1588 hardware timer.
//!
//! This function allows an application to register a handler for all
//! interrupts generated by the IEEE-1588 hardware timer in the Ethernet MAC.
//! To allow minimal latency timer handling, the callback function provided
//! will be called in interrupt context, regardless of whether or not lwIP is
//! configured to operate with an RTOS.  In an RTOS environment, the callback
//! function is responsible for ensuring that all processing it performs is
//! compatible with the low level interrupt context it is called in.
//!
//! The callback function takes two parameters.  The first is the base address
//! of the MAC reporting the timer interrupt and the second is the timer
//! interrupt status as reported by EMACTimestampIntStatus().  Note that
//! EMACTimestampIntStatus() causes the timer interrupt sources to be cleared
//! so the application should not call EMACTimestampIntStatus() within the
//! handler.
//!
//! \return None.
//
```

```
//*****************************************************************************
void
lwIPTimerCallbackRegister(tHardwareTimerHandler pfnTimerFunc)
{
    //
    // Remember the callback function address passed.
    //
    g_pfnTimerHandler = pfnTimerFunc;
}


//*****************************************************************************
//
//! Handles Ethernet interrupts for the lwIP TCP/IP stack.
//!
//! This function handles Ethernet interrupts for the lwIP TCP/IP stack.  At
//! the lowest level, all receive packets are placed into a packet queue for
//! processing at a higher level.  Also, the transmit packet queue is checked
//! and packets are drained and transmitted through the Ethernet MAC as needed.
//! If the system is configured without an RTOS, additional processing is
//! performed at the interrupt level.  The packet queues are processed by the
//! lwIP TCP/IP code, and lwIP periodic timers are serviced (as needed).
//!
//! \return None.
//
//*****************************************************************************
void
lwIPEthernetIntHandler(void)
{
    uint32_t ui32Status;
    uint32_t ui32TimerStatus;
#if !NO_SYS
    portBASE_TYPE xWake;
#endif

    //
    // Read and Clear the interrupt.
    //
    ui32Status = MAP_EMACIntStatus(EMAC0_BASE, true);

    //
    // If the interrupt really came from the Ethernet and not our
    // timer, clear it.
    //
    if(ui32Status)
    {
        MAP_EMACIntClear(EMAC0_BASE, ui32Status);
    }

    //
    // Check to see whether a hardware timer interrupt has been reported.
    //
    if(ui32Status & EMAC_INT_TIMESTAMP)
    {
        //
        // Yes - read and clear the timestamp interrupt status.
        //
        ui32TimerStatus = EMACTimestampIntStatus(EMAC0_BASE);

        //
        // If a timer interrupt handler has been registered, call it.
        //
        if(g_pfnTimerHandler)
        {
            g_pfnTimerHandler(EMAC0_BASE, ui32TimerStatus);
        }
    }

    //
    // The handling of the interrupt is different based on the use of a RTOS.
    // A RTOS is being used.  Signal the Ethernet interrupt task.
    //
    xQueueSendFromISR(g_pInterrupt, (void *)&ui32Status, &xWake);

    //
    // Disable the Ethernet interrupts.  Since the interrupts have not been
    // handled, they are not asserted.  Once they are handled by the Ethernet
    // interrupt task, it will re-enable the interrupts.
    //
    MAP_EMACIntDisable(EMAC0_BASE, (EMAC_INT_RECEIVE | EMAC_INT_TRANSMIT |
                                    EMAC_INT_TX_STOPPED |
                                    EMAC_INT_RX_NO_BUFFER |
                                    EMAC_INT_RX_STOPPED | EMAC_INT_PHY));

    //
    // Potentially task switch as a result of the above queue write.
    //
#if RTOS_FREERTOS
    if(xWake == pdTRUE)
    {
        portYIELD_FROM_ISR(true);
    }
#endif
}

//*****************************************************************************
//
//! Returns the IP address for this interface.
//!
//! This function will read and return the currently assigned IP address for
//! the Stellaris Ethernet interface.
//!
//! \return Returns the assigned IP address for this interface.
//
//*****************************************************************************
uint32_t
lwIPLocalIPAddrGet(void)
{
#if LWIP_AUTOIP || LWIP_DHCP
    if(g_bLinkActive)
    {
        return((uint32_t)g_sNetIF.ip_addr.addr);
    }
    else
    {
        return(0xffffffff);
    }
#else
    return((uint32_t)g_sNetIF.ip_addr.addr);
#endif
}

//*****************************************************************************
//
//! Returns the network mask for this interface.
//!
//! This function will read and return the currently assigned network mask for
//! the Stellaris Ethernet interface.
//!
//! \return the assigned network mask for this interface.
//
//*****************************************************************************
uint32_t
lwIPLocalNetMaskGet(void)
{
    return((uint32_t)g_sNetIF.netmask.addr);
}

//*****************************************************************************
//
//! Returns the gateway address for this interface.
//!
//! This function will read and return the currently assigned gateway address
//! for the Stellaris Ethernet interface.
//!
//! \return the assigned gateway address for this interface.
```

```
//
//*****************************************************************************
uint32_t
lwIPLocalGWAddrGet(void)
{
    return((uint32_t)g_sNetIF.gw.addr);
}

//*****************************************************************************
//
//! Returns the local MAC/HW address for this interface.
//!
//! \param pui8MAC is a pointer to an array of bytes used to store the MAC
//! address.
//!
//! This function will read the currently assigned MAC address into the array
//! passed in \e pui8MAC.
//!
//! \return None.
//
//*****************************************************************************
void
lwIPLocalMACGet(uint8_t *pui8MAC)
{
    MAP_EMACAddrGet(EMAC0_BASE, 0, pui8MAC);
}

//*****************************************************************************
//
// Completes the network configuration change.  This is directly called when
// not using a RTOS and provided as a callback to the TCP/IP thread when using
// a RTOS.
//
//*****************************************************************************
static void
lwIPPrivateNetworkConfigChange(void *pvArg)
{
    uint32_t ui32IPMode;
    struct ip_addr ip_addr;
    struct ip_addr net_mask;
    struct ip_addr gw_addr;

    //
    // Get the new address mode.
    //
    ui32IPMode = (uint32_t)pvArg;

    //
    // Setup the network address values.
    //
    if(ui32IPMode == IPADDR_USE_STATIC)
    {
        ip_addr.addr = htonl(g_ui32IPAddr);
        net_mask.addr = htonl(g_ui32NetMask);
        gw_addr.addr = htonl(g_ui32GWAddr);
    }
#if LWIP_DHCP || LWIP_AUTOIP
    else
    {
        ip_addr.addr = 0;
        net_mask.addr = 0;
        gw_addr.addr = 0;
    }
#endif

    //
    // Switch on the current IP Address Aquisition mode.
    //
    switch(g_ui32IPMode)
    {
        //
        // Static IP
        //
        case IPADDR_USE_STATIC:
        {
            //
            // Set the new address parameters.  This will change the address
            // configuration in lwIP, and if necessary, will reset any links
            // that are active.  This is valid for all three modes.
            //
            netif_set_addr(&g_sNetIF, &ip_addr, &net_mask, &gw_addr);

            //
            // If we are going to DHCP mode, then start the DHCP server now.
            //
#if LWIP_DHCP
            if((ui32IPMode == IPADDR_USE_DHCP) && g_bLinkActive)
            {
                dhcp_start(&g_sNetIF);
            }
#endif

            //
            // If we are going to AutoIP mode, then start the AutoIP process
            // now.
            //
#if LWIP_AUTOIP
            if((ui32IPMode == IPADDR_USE_AUTOIP) && g_bLinkActive)
            {
                autoip_start(&g_sNetIF);
            }
#endif

            //
            // And we're done.
            //
            break;
        }

        //
        // DHCP (with AutoIP fallback).
        //
#if LWIP_DHCP
        case IPADDR_USE_DHCP:
        {
            //
            // If we are going to static IP addressing, then disable DHCP and
            // force the new static IP address.
            //
            if(ui32IPMode == IPADDR_USE_STATIC)
            {
                dhcp_stop(&g_sNetIF);
                netif_set_addr(&g_sNetIF, &ip_addr, &net_mask, &gw_addr);
            }

            //
            // If we are going to AUTO IP addressing, then disable DHCP, set
            // the default addresses, and start AutoIP.
            //
#if LWIP_AUTOIP
            else if(ui32IPMode == IPADDR_USE_AUTOIP)
            {
                dhcp_stop(&g_sNetIF);
                netif_set_addr(&g_sNetIF, &ip_addr, &net_mask, &gw_addr);
                if(g_bLinkActive)
                {
                    autoip_start(&g_sNetIF);
                }
            }
#endif
            break;
        }
#endif

        //
        // AUTOIP
        //
#if LWIP_AUTOIP
        case IPADDR_USE_AUTOIP:
        {
            //
            // If we are going to static IP addressing, then disable AutoIP and
            // force the new static IP address.
            //
            if(ui32IPMode == IPADDR_USE_STATIC)
            {
                autoip_stop(&g_sNetIF);
                netif_set_addr(&g_sNetIF, &ip_addr, &net_mask, &gw_addr);
            }

            //
            // If we are going to DHCP addressing, then disable AutoIP, set the
            // default addresses, and start dhcp.
            //
#if LWIP_DHCP
            else if(ui32IPMode == IPADDR_USE_DHCP)
            {
                autoip_stop(&g_sNetIF);
                netif_set_addr(&g_sNetIF, &ip_addr, &net_mask, &gw_addr);
                if(g_bLinkActive)
                {
                    dhcp_start(&g_sNetIF);
                }
            }
#endif
            break;
        }
#endif
    }

    //
    // Bring the interface up.
    //
    netif_set_up(&g_sNetIF);

    //
    // Save the new mode.
    //
    g_ui32IPMode = ui32IPMode;
}

//*****************************************************************************
//
//! Change the configuration of the lwIP network interface.
//!
//! \param ui32IPAddr is the new IP address to be used (static).
//! \param ui32NetMask is the new network mask to be used (static).
//! \param ui32GWAddr is the new Gateway address to be used (static).
//! \param ui32IPMode is the IP Address Mode.  \b IPADDR_USE_STATIC 0 will
//! force static IP addressing to be used, \b IPADDR_USE_DHCP will force DHCP
//! with fallback to Link Local (Auto IP), while \b IPADDR_USE_AUTOIP will
//! force Link Local only.
//!
//! This function will evaluate the new configuration data.  If necessary, the
//! interface will be brought down, reconfigured, and then brought back up
//! with the new configuration.
//!
//! \return None.
//
//*****************************************************************************
void
lwIPNetworkConfigChange(uint32_t ui32IPAddr, uint32_t ui32NetMask,
                        uint32_t ui32GWAddr, uint32_t ui32IPMode)
{
    //
    // Check the parameters.
    //
#if LWIP_DHCP && LWIP_AUTOIP
    ASSERT((ui32IPMode == IPADDR_USE_STATIC) ||
            (ui32IPMode == IPADDR_USE_DHCP) ||
            (ui32IPMode == IPADDR_USE_AUTOIP));
#elif LWIP_DHCP
    ASSERT((ui32IPMode == IPADDR_USE_STATIC) ||
            (ui32IPMode == IPADDR_USE_DHCP));
#elif LWIP_AUTOIP
    ASSERT((ui32IPMode == IPADDR_USE_STATIC) ||
            (ui32IPMode == IPADDR_USE_AUTOIP));
#else
    ASSERT(ui32IPMode == IPADDR_USE_STATIC);
#endif

    //
    // Save the network configuration for later use by the private network
    // configuration change.
    //
    g_ui32IPAddr = ui32IPAddr;
    g_ui32NetMask = ui32NetMask;
    g_ui32GWAddr = ui32GWAddr;

    //
    // Complete the network configuration change.  The remainder is done
    // immediately if not using a RTOS and it is deferred to the TCP/IP
    // thread's context if using a RTOS.
    //
    tcpip_callback(lwIPPrivateNetworkConfigChange, (void *)ui32IPMode);
}

//*****************************************************************************
//
// Close the Doxygen group.
//! @}
//
//*****************************************************************************
```

code/asv_ap_mcu/src/tm4c129.ld

```
_stack_size = 4K;

MEMORY
{
    FLASH (rx) : ORIGIN = 0x00000000, LENGTH = 1024K
    SRAM (rwx) : ORIGIN = 0x20000000, LENGTH = 256K
}

SECTIONS
{
    .text :
    {
        _start_ext = .;
        KEEP(*(.isr_vector))
        *(.text*)
        *(.rodata*)
        _end_text = .;
    } > FLASH

    .data :
    {
        _start_data = .;
        *(vtable)
        *(.data*)
        _end_data = .;

    } > SRAM AT > FLASH

    .ARM.extab   : { *(.ARM.extab* .gnu.linkonce.armextab.*) } > FLASH
    ___exidx_start = .;
    .ARM.exidx   : { *(.ARM.exidx* .gnu.linkonce.armexidx.*) } > FLASH
    ___exidx_end = .;


    .bss :
```

```
    {
        _start_bss = .;
        *(.bss*)
        *(COMMON)
        _end_bss = .;
    } > SRAM

    _heap_bottom = .;
    _heap_top = ORIGIN(SRAM) + LENGTH(SRAM) - _stack_size;

    _stack_bottom = _heap_top;
    _stack_top = ORIGIN(SRAM) + LENGTH(SRAM);

}
```

code/asv_ap_mcu/src/startup.c

```
/*
* Copyright (c) 2012, Mauro Scomparin
* All rights reserved.
*
* Redistribution and use in source and binary forms, with or without
* modification, are permitted provided that the following conditions are met:
*     * Redistributions of source code must retain the above copyright
*       notice, this list of conditions and the following disclaimer.
*     * Redistributions in binary form must reproduce the above copyright
*       notice, this list of conditions and the following disclaimer in the
*       documentation and/or other materials provided with the distribution.
*     * Neither the name of Mauro Scomparin nor the
*       names of its contributors may be used to endorse or promote products
*       derived from this software without specific prior written permission.
*
* THIS SOFTWARE IS PROVIDED BY Mauro Scomparin ''AS IS'' AND ANY
* EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
* WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
* DISCLAIMED. IN NO EVENT SHALL Mauro Scomparin BE LIABLE FOR ANY
* DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
* (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
* File:          LM4F_startup.c.
* Author:        Mauro Scomparin <http://scompoprojects.worpress.com>.
* Version:       1.0.0.
* Description:   LM4F120H5QR startup code.
*/

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_nvic.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/rom.h"
#include "driverlib/gpio.h"

//-----------------------------------------------------------------------------
//                              Functions declarations
//-----------------------------------------------------------------------------

// Main should be defined on your main file so it's extern.
extern int main(void);
// rst_handler contains the code to run on reset.
void rst_handler(void);
// nmi_handler it's the code for an non maskerable interrupt.
void nmi_handler(void);
// this is just the default handler.
void empty_def_handler(void);
// this is the code for an hard fault.
void hardfault_handler(void);

```

```
// External declaration for the interrupt handler used by the application.
extern void lwIPEthernetIntHandler(void);
extern void xPortPendSVHandler(void);
extern void vPortSVCHandler(void);
extern void xPortSysTickHandler(void);
extern void UART2_int_handler(void);
extern void UART4_int_handler(void);
extern void UART5_int_handler(void);
extern void UART7_int_handler(void);

//---------------------------------------------------------------------------
//                          Variables declarations
//---------------------------------------------------------------------------

// defined by the linker it's the stack top variable (End of ram)
extern unsigned long _stack_top;
//extern uint32_t _stack_top;  // Defined in the linker script
// defined by the liker, this are just start and end marker for each section.
// .text (code)
extern unsigned long _start_text;
extern unsigned long _end_text;
// .data (data to be copied on ram)
extern unsigned long _start_data;
extern unsigned long _end_data;
// .bss (uninitialized data to set to 0);
extern unsigned long _start_bss;
extern unsigned long _end_bss;

// NVIC ISR table
// the funny looking void(* myvectors[])(void) basically it's a way to make cc accept an array of function pointers.
// The vector table. Note that the proper constructs must be placed on this to
// ensure that it ends up at physical address 0x0000.0000.
__attribute__ ((section(".isr_vector")))
void(* myvectors[])(void) = {
    // This are the fixed priority interrupts and the stack pointer loaded at startup at R13 (SP).
    //                                            VECTOR N (Check Datasheet)
    // here the compiler it's boring.. have to figure that out
    //(void (*)(void))((uint32_t) &_stack_top),
    (void (*)(void))((unsigned long) &_stack_top),
    //(void (*)) &_stack_top,
                            // stack pointer should be
                            // placed here at startup.          0
    rst_handler,                              // The reset handler
    nmi_handler,                                 // The NMI handler
    hardfault_handler,                              // The hard fault handler
    empty_def_handler,                     // The MPU fault handler
    empty_def_handler,                     // The bus fault handler
    empty_def_handler,                     // The usage fault handler
    0,                                     // Reserved
    0,                                     // Reserved
    0,                                     // Reserved
    0,                                     // Reserved
    vPortSVCHandler,                       // SVCall handler
    empty_def_handler,                     // Debug monitor handler
    0,                                     // Reserved
    xPortPendSVHandler,                    // The PendSV handler
    xPortSysTickHandler,                   // The SysTick handler
    empty_def_handler,                     // GPIO Port A
    empty_def_handler,                     // GPIO Port B
    empty_def_handler,                     // GPIO Port C
    empty_def_handler,                     // GPIO Port D
    empty_def_handler,                     // GPIO Port E
    empty_def_handler,                     // UART0 Rx and Tx
    empty_def_handler,                     // UART1 Rx and Tx
    empty_def_handler,                     // SSI0 Rx and Tx
    empty_def_handler,                     // I2C0 Master and Slave
    empty_def_handler,                     // PWM Fault
    empty_def_handler,                     // PWM Generator 0
    empty_def_handler,                     // PWM Generator 1
    empty_def_handler,                     // PWM Generator 2
    empty_def_handler,                     // Quadrature Encoder 0
    empty_def_handler,                     // ADC Sequence 0
    empty_def_handler,                     // ADC Sequence 1
    empty_def_handler,                     // ADC Sequence 2
    empty_def_handler,                     // ADC Sequence 3
    empty_def_handler,                     // Watchdog timer
    empty_def_handler,                     // Timer 0 subtimer A
    empty_def_handler,                     // Timer 0 subtimer B
    empty_def_handler,                     // Timer 1 subtimer A
    empty_def_handler,                     // Timer 1 subtimer B
    empty_def_handler,                     // Timer 2 subtimer A
    empty_def_handler,                     // Timer 2 subtimer B
    empty_def_handler,                     // Analog Comparator 0
    empty_def_handler,                     // Analog Comparator 1
    empty_def_handler,                     // Analog Comparator 2
    empty_def_handler,                     // System Control (PLL, OSC, BO)
    empty_def_handler,                     // FLASH Control
    empty_def_handler,                     // GPIO Port F
    empty_def_handler,                     // GPIO Port G
    empty_def_handler,                     // GPIO Port H
    UART2_int_handler,                     // UART2 Rx and Tx
    empty_def_handler,                     // SSI1 Rx and Tx
    empty_def_handler,                     // Timer 3 subtimer A
    empty_def_handler,                     // Timer 3 subtimer B
    empty_def_handler,                     // I2C1 Master and Slave
    empty_def_handler,                     // CAN0
    empty_def_handler,                     // CAN1
    lwIPEthernetIntHandler,                // Ethernet
    empty_def_handler,                     // Hibernate
    empty_def_handler,                     // USB0
    empty_def_handler,                     // PWM Generator 3
    empty_def_handler,                     // uDMA Software Transfer
    empty_def_handler,                     // uDMA Error
    empty_def_handler,                     // ADC1 Sequence 0
    empty_def_handler,                     // ADC1 Sequence 1
    empty_def_handler,                     // ADC1 Sequence 2
    empty_def_handler,                     // ADC1 Sequence 3
    empty_def_handler,                     // External Bus Interface 0
    empty_def_handler,                     // GPIO Port J
    empty_def_handler,                     // GPIO Port K
    empty_def_handler,                     // GPIO Port L
    empty_def_handler,                     // SSI2 Rx and Tx
    empty_def_handler,                     // SSI3 Rx and Tx
    empty_def_handler,                     // UART3 Rx and Tx
    UART4_int_handler,                     // UART4 Rx and Tx
    UART5_int_handler,                     // UART5 Rx and Tx
    empty_def_handler,                     // UART6 Rx and Tx
    UART7_int_handler,                     // UART7 Rx and Tx
    empty_def_handler,                     // I2C2 Master and Slave
    empty_def_handler,                     // I2C3 Master and Slave
    empty_def_handler,                     // Timer 4 subtimer A
    empty_def_handler,                     // Timer 4 subtimer B
    empty_def_handler,                     // Timer 5 subtimer A
    empty_def_handler,                     // Timer 5 subtimer B
    empty_def_handler,                     // FPU
    0,                                     // Reserved
    0,                                     // Reserved
    empty_def_handler,                     // I2C4 Master and Slave
    empty_def_handler,                     // I2C5 Master and Slave
    empty_def_handler,                     // GPIO Port M
    empty_def_handler,                     // GPIO Port N
    0,                                     // Reserved
    empty_def_handler,                     // Tamper
    empty_def_handler,                     // GPIO Port P (Summary or P0)
    empty_def_handler,                     // GPIO Port P1
    empty_def_handler,                     // GPIO Port P2
    empty_def_handler,                     // GPIO Port P3
    empty_def_handler,                     // GPIO Port P4
    empty_def_handler,                     // GPIO Port P5
    empty_def_handler,                     // GPIO Port P6
    empty_def_handler,                     // GPIO Port P7
    empty_def_handler,                     // GPIO Port Q (Summary or Q0)
    empty_def_handler,                     // GPIO Port Q1
    empty_def_handler,                     // GPIO Port Q2
    empty_def_handler,                      // GPIO Port Q3
    empty_def_handler,                      // GPIO Port Q4
    empty_def_handler,                      // GPIO Port Q5
    empty_def_handler,                      // GPIO Port Q6
    empty_def_handler,                      // GPIO Port Q7
    empty_def_handler,                      // GPIO Port R
    empty_def_handler,                      // GPIO Port S
    empty_def_handler,                      // SHA/MD5 0
    empty_def_handler,                      // AES 0
    empty_def_handler,                      // DES3DES 0
    empty_def_handler,                      // LCD Controller 0
    empty_def_handler,                      // Timer 6 subtimer A
    empty_def_handler,                      // Timer 6 subtimer B
    empty_def_handler,                      // Timer 7 subtimer A
    empty_def_handler,                      // Timer 7 subtimer B
    empty_def_handler,                      // I2C6 Master and Slave
    empty_def_handler,                      // I2C7 Master and Slave
    empty_def_handler,                      // HIM Scan Matrix Keyboard 0
    empty_def_handler,                      // One Wire 0
    empty_def_handler,                      // HIM PS/2 0
    empty_def_handler,                      // HIM LED Sequencer 0
    empty_def_handler,                      // HIM Consumer IR 0
    empty_def_handler,                      // I2C8 Master and Slave
    empty_def_handler,                      // I2C9 Master and Slave
    empty_def_handler                       // GPIO Port T
};

//-----------------------------------------------------------------------------
//                          Function implementations.
//-----------------------------------------------------------------------------

/*
* System on reset code. NVIC 1
* Here I prepare the memory for the c compiler.
* The stack pointer should be set at the beginning with the NVIC table already.
* Copy the .data segment from flash into ram.
* 0 to the .bss segment
*/

void rst_handler(void){
    // Copy the .data section pointers to ram from flash.
    // Look at LD manual (Optional Section Attributes).

    // source and destination pointers
    unsigned long *src;
    unsigned long *dest;

    //this should be good!
    src = &_end_text;
    dest = &_start_data;

    //this too
    while(dest < &_end_data)
    {
        *dest++ = *src++;
    }

    // now set the .bss segment to 0!
    dest = &_start_bss;
    while(dest < &_end_bss){
        *dest++ = 0;
    }

    // after setting copying .data to ram and "zero-ing" .bss we are good
    // to start the main() method!
    // There you go!
    main();
}

// NMI Exception handler code NVIC 2
void nmi_handler(void){
    ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, 0); // Red
    // Just loop forever, so if you want to debug the processor it's running.
    while(1){
    }
}

// Hard fault handler code NVIC 3
void hardfault_handler(void){
    ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, 0); // Red
    // Just loop forever, so if you want to debug the processor it's running.
    while(1){
    }
}

// Empty handler used as default.
void empty_def_handler(void){
    ROM_GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_0, 0); // Red
    //ROM_GPIOPinWrite(GPIO_PORTH_BASE, GPIO_PIN_2, 0);
    // Just loop forever, so if you want to debug the processor it's running.
    while(1){
    }
}
```

code/asv_ap_mcu/src/syscalls.c

```
#include <sys/types.h>          //Needed for caddr_t
#include <stdint.h>

#include "inc/hw_memmap.h"      //Needed for GPIO Pins/UART base
#include "inc/hw_types.h"       //Needed for SysTick Types
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"

char *heap_end = 0;

caddr_t _sbrk(unsigned int incr) {

    extern unsigned long _heap_bottom;
    extern unsigned long _heap_top;
    static char *prev_heap_end;

    if (heap_end == 0) {
        heap_end = (caddr_t)&_heap_bottom;
    }

    prev_heap_end = heap_end;

    if (heap_end + incr > (caddr_t)&_heap_top) {
        return (caddr_t)0;
    }

    heap_end += incr;

    return (caddr_t) prev_heap_end;
}

int _close(int file) {
    return -1;
}

int _fstat(int file) {
    return -1;
}

int _isatty(int file) {
    return -1;
}

int _lseek(int file, int ptr, int dir) {
    return -1;
}

int _open(const char *name, int flags, int mode) {
    return -1;
}

int _read(int file, char *ptr, int len) {

    unsigned int i;
    for( i = 0; i < len; i++ ){
        ptr[i] = (char)ROM_UARTCharGet(UART5_BASE);
    }
    return len;
}

int _write(int file, char *ptr, unsigned int len) {

    unsigned int i;
    for(i = 0; i < len; i++){
        ROM_UARTCharPut(UART5_BASE, ptr[i]);
    }
    return i;
}
```

code/asv_ap_mcu/src/Makefile

```
# Copyright (c) 2012, Mauro Scomparin
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#     * Redistributions of source code must retain the above copyright
#       notice, this list of conditions and the following disclaimer.
#     * Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#     * Neither the name of Mauro Scomparin nor the
#       names of its contributors may be used to endorse or promote products
#       derived from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY Mauro Scomparin ``AS IS'' AND ANY
# EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
# WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL Mauro Scomparin BE LIABLE FOR ANY
# DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
# (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
# ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
# (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
# SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
# File:          Makefile.
# Author:        Mauro Scomparin <http://scompoprojects.worpress.com>.
# Version:       1.0.1.
# Description:   Sample makefile, modified version inspired by Wollw @ github
#
# Modified by:   Joakim Myrland
# website:       www.LDA.as
# email:         joakim.myrland@LDA.as
# project:       https://github.com/Lindem-Data-Acquisition-AS/iot_tiva_template/
#

#global names
LIB_PORT_NAME    = tm4c129
PROJECT_NAME     = iot_template_$(LIB_PORT_NAME)

#path names relative to makefile:
ROOT                    = ..
#path names relative to ROOT:
SRC_PATH                = src
BIN_PATH                = bin
TIVAWARE_PATH           = iot_mcu_lib/TivaWare_C_Series-2.1.0.12573
FreeRTOS_PATH           = iot_mcu_lib/FreeRTOSV8.2.0
lwIP_PATH               = iot_mcu_lib/lwip-1.4.1
jsmn_PATH               = iot_mcu_lib/jsmn
```

```
#compiler setup
COMPILER        = gcc
TOOLCHAIN       = arm-none-eabi
PART            = TM4C1294NCPDT
#TM4C1294KCPDT
#TM4C1294NCPDT
#TM4C129ENCPDT
CPU             = cortex-m4
FPU             = fpv4-sp-d16
FABI            = softfp

LINKER_FILE         = ${LIB_PORT_NAME}.ld
STARTUP_FILE        = main

CC = ${TOOLCHAIN}-gcc
LD = ${TOOLCHAIN}-ld
CP = ${TOOLCHAIN}-objcopy
OD = ${TOOLCHAIN}-objdump

CFLAGS  = -mthumb -mcpu=$(CPU) -mfpu=$(FPU) -mfloat-abi=$(FABI)
CFLAGS += -Os -ffunction-sections -fdata-sections
CFLAGS += -MD -std=gnu99 -Wall
CFLAGS += -DPART_$(PART) -c -DTARGET_IS_TM4C129_RA0
CFLAGS += -c -g
#CFLAGS += -DDEBUG
CFLAGS += -DRTOS_FREERTOS


#include header directories
CFLAGS += -I ${ROOT}
CFLAGS += -I ${ROOT}/${SRC_PATH}
CFLAGS += -I ${ROOT}/${SRC_PATH}/config
CFLAGS += -I ${ROOT}/${SRC_PATH}/tasks
CFLAGS += -I ${ROOT}/${TIVAWARE_PATH}
#include FreeRTOS
CFLAGS += -I ${ROOT}/${FreeRTOS_PATH}/FreeRTOS/Source/portable/GCC/ARM_CM4F
CFLAGS += -I ${ROOT}/${FreeRTOS_PATH}/FreeRTOS/Source/include
#include lwIP
CFLAGS += -I ${ROOT}/${lwIP_PATH}/ports/${LIB_PORT_NAME}/include
CFLAGS += -I ${ROOT}/${lwIP_PATH}/src/include
CFLAGS += -I ${ROOT}/${lwIP_PATH}/src/include/ipv4
CFLAGS += -I ${ROOT}/${lwIP_PATH}/src/include/netif
#include lwIP apps
CFLAGS += -I ${ROOT}/${lwIP_PATH}/apps
#include jsmn
CFLAGS += -I ${ROOT}/${jsmn_PATH}

#include source directories/files
SRC += $(wildcard ${ROOT}/${SRC_PATH}/*.c)
SRC += $(wildcard ${ROOT}/${SRC_PATH}/asv_ap/*.c)
SRC += $(wildcard ${ROOT}/${SRC_PATH}/tasks/*.c)
#include FreeRTOS
SRC += $(wildcard ${ROOT}/${FreeRTOS_PATH}/FreeRTOS/Source/portable/GCC/ARM_CM4F/*.c)
SRC += $(wildcard ${ROOT}/${FreeRTOS_PATH}/FreeRTOS/Source/portable/MemMang/heap_2.c)
SRC += $(wildcard ${ROOT}/${FreeRTOS_PATH}/FreeRTOS/Source/*.c)
#include lwIP
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/ports/${LIB_PORT_NAME}/*.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/ports/${LIB_PORT_NAME}/netif/*.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/src/core/*.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/src/core/ipv4/*.c)
#SRC += $(wildcard ${ROOT}/${lwIP_PATH}/src/core/ipv6/*.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/src/core/snmp/*.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/src/netif/*.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/src/netif/ppp/*.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/src/api/*.c)
#include lwIP apps
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/apps/httpserver_raw/httpd.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/apps/httpserver_raw/httpd_post.c)
SRC += $(wildcard ${ROOT}/${lwIP_PATH}/apps/httpserver_raw/fs.c)

#include jsmn
SRC += $(wildcard ${ROOT}/${jsmn_PATH}/jsmn.c)
```

```
LIB_GCC_PATH     = ${shell ${CC} ${CFLAGS} -print-libgcc-file-name}
LIBC_PATH        = ${shell ${CC} ${CFLAGS} -print-file-name=libc.a}
LIBM_PATH        = ${shell ${CC} ${CFLAGS} -print-file-name=libm.a}
LFLAGS           = --gc-sections
CPFLAGS          = -Obinary
ODFLAGS          = -S

OBJS = $(SRC:.c=.o)
D_FILES = $(SRC:.c=.d)

#this ECHO functions prints compile progress [nn%] to stdout
ifndef ECHO
T := $(shell $(MAKE) $(MAKECMDGOALS) --no-print-directory \
      -nrRf $(firstword $(MAKEFILE_LIST)) \
      ECHO="COUNTTHIS" | grep -c "COUNTTHIS")

N := x
C = $(words $N)$(eval N := x $N)
ECHO = echo "`expr "   \`expr $C '*' 100 / $T\`" : '.*\(....\)$$'`%"
endif

#rules
all: $(OBJS) ${PROJECT_NAME}.axf ${PROJECT_NAME}
	size ${ROOT}/${BIN_PATH}/${PROJECT_NAME}.axf
	@echo make complete

%.o: %.c
#@echo .
	@mkdir -p ${ROOT}/${BIN_PATH}
	@$(CC) -c $(CFLAGS) $< -o $@
	@$(ECHO)

${PROJECT_NAME}.axf: $(OBJS)
	@$(LD) -T $(LINKER_FILE) $(LFLAGS) -o ${ROOT}/${BIN_PATH}/${PROJECT_NAME}.axf $(OBJS) $(LIBM_PATH) $(LIBC_PATH) $(LIB_GCC_PATH) ${ROOT}/${TIVAWARE_PATH}/driverlib/${COMPILER}/libdriver.a

${PROJECT_NAME}: ${PROJECT_NAME}.axf
	@$(CP) $(CPFLAGS) ${ROOT}/${BIN_PATH}/${PROJECT_NAME}.axf ${ROOT}/${BIN_PATH}/${PROJECT_NAME}.bin
	@$(OD) $(ODFLAGS) ${ROOT}/${BIN_PATH}/${PROJECT_NAME}.axf > ${ROOT}/${BIN_PATH}/${PROJECT_NAME}.lst

# make clean rule
clean:
	@rm -fr ${ROOT}/${BIN_PATH}
	@rm -f ${OBJS}
	@rm -f ${D_FILES}
	@echo clean complete

# Rule to flash the project to the board
flash: all
	openocd \
	-c "source [find interface/ftdi/olimex-arm-usb-ocd-h.cfg]" \
	-c "set WORKAREASIZE 0x40000" \
	-c "set CHIPNAME TM4C1294NCPDT" \
	-c "source [find target/stellaris.cfg]" \
	-c "init" \
	-c "targets" \
	-c "reset halt" \
	-c "flash info 0" \
	-c "flash banks" \
	-c "flash write_image erase $(ROOT)/$(BIN_PATH)/$(PROJECT_NAME).axf" \
	-c "verify_image $(ROOT)/$(BIN_PATH)/$(PROJECT_NAME).axf" \
	-c "reset" \
	-c "shutdown"

debug: flash
	arm-none-eabi-gdb -ex \
	'target extended-remote | openocd \
	-c "source [find interface/ftdi/olimex-arm-usb-ocd-h.cfg]" \
	-c "set WORKAREASIZE 0x40000" \
	-c "set CHIPNAME TM4C1294NCPDT" \
    -c "transport select jtag" \
    -c "source [find target/stellaris.cfg]" \
    "gdb_port pipe; log_output openocd.log"; monitor reset halt'\
     $(ROOT)/$(BIN_PATH)/$(PROJECT_NAME).axf
```

Appendix N

Code for ASV control panel

Server and client-side JavaScript code for the ASV control panel, as described in section 3.3.9. The code is licensed under the MIT license (listing D.1).

code/ASV_control_panel/server.js

```
/* Setup logging */
var winston = require('winston');
winston.add(winston.transports.File, { filename: 'nodejs.log' });

winston.info('Starting␣server.');

/* Setting up webserver */
var node_static = require('node-static');

var files = new node_static.Server('./public');

function http_handler(request, response) {
    request.on('end', function() {
        files.serve(request, response);
    }).resume();
}

var http = require('http');

var app = http.createServer(http_handler);
app.listen(80, '0.0.0.0');

/* Setting up socket.io */
var io = require('socket.io').listen(app);

io.sockets.on('connection', function(socket) {

    socket.on('message', function(data) {

        //console.log("sending:" + data);
        winston.info("sending:" + data);

        var client = new net.Socket();
        var message = new Buffer(data);

        client.connect(30470, "192.168.1.21", function() {
            client.write(message);
            client.destroy();
        });

    });
});

/* Setting up TCP client */
var net = require('net');
```

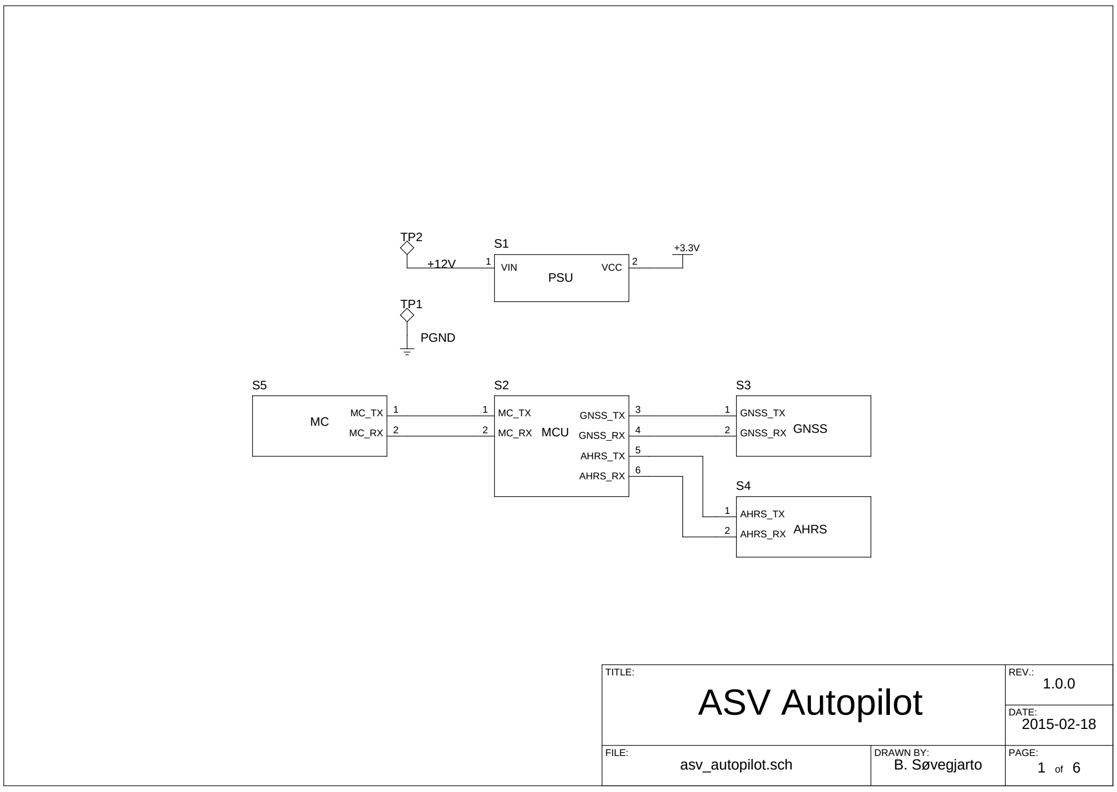
```

/* Setting up UDP server */
var dgram = require('dgram');

var udp_server = dgram.createSocket("udp4");

udp_server.on("listening", function() {
    var address = udp_server.address();
    //console.log("Server listening on " + address.address + ":" + address.port);
    winston.info("Server␣listening␣on␣" + address.address + ":" + address.port);
});

udp_server.on("message", function(msg, rinfo) {
    //console.log("Server got: " + msg + " from " + rinfo.address + ":" + rinfo.port);
    winston.info("Server␣got:␣" + msg + "␣from␣" + rinfo.address + ":" + rinfo.port);
    io.sockets.emit('message', "" + msg);
});

udp_server.on('error', function(err) {
    console.error(err);
    process.exit(0);
});

udp_server.bind(8000);
```

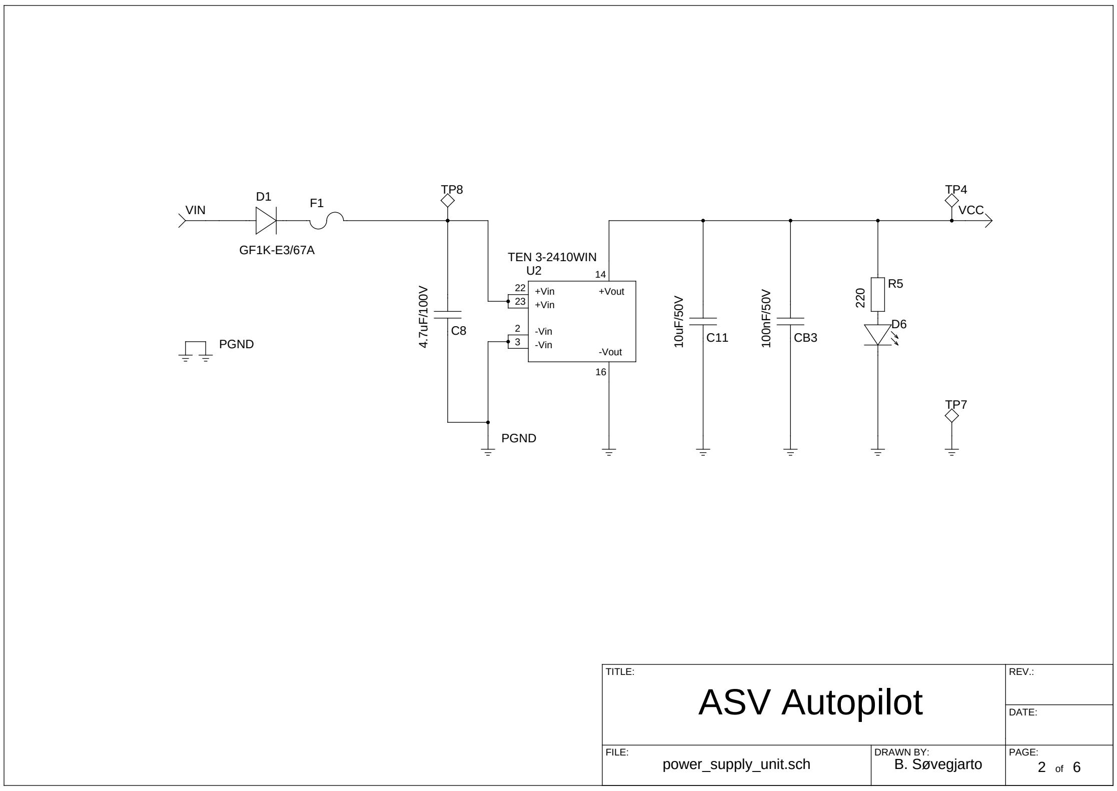
code/ASV_control_panel/public/js/application.js

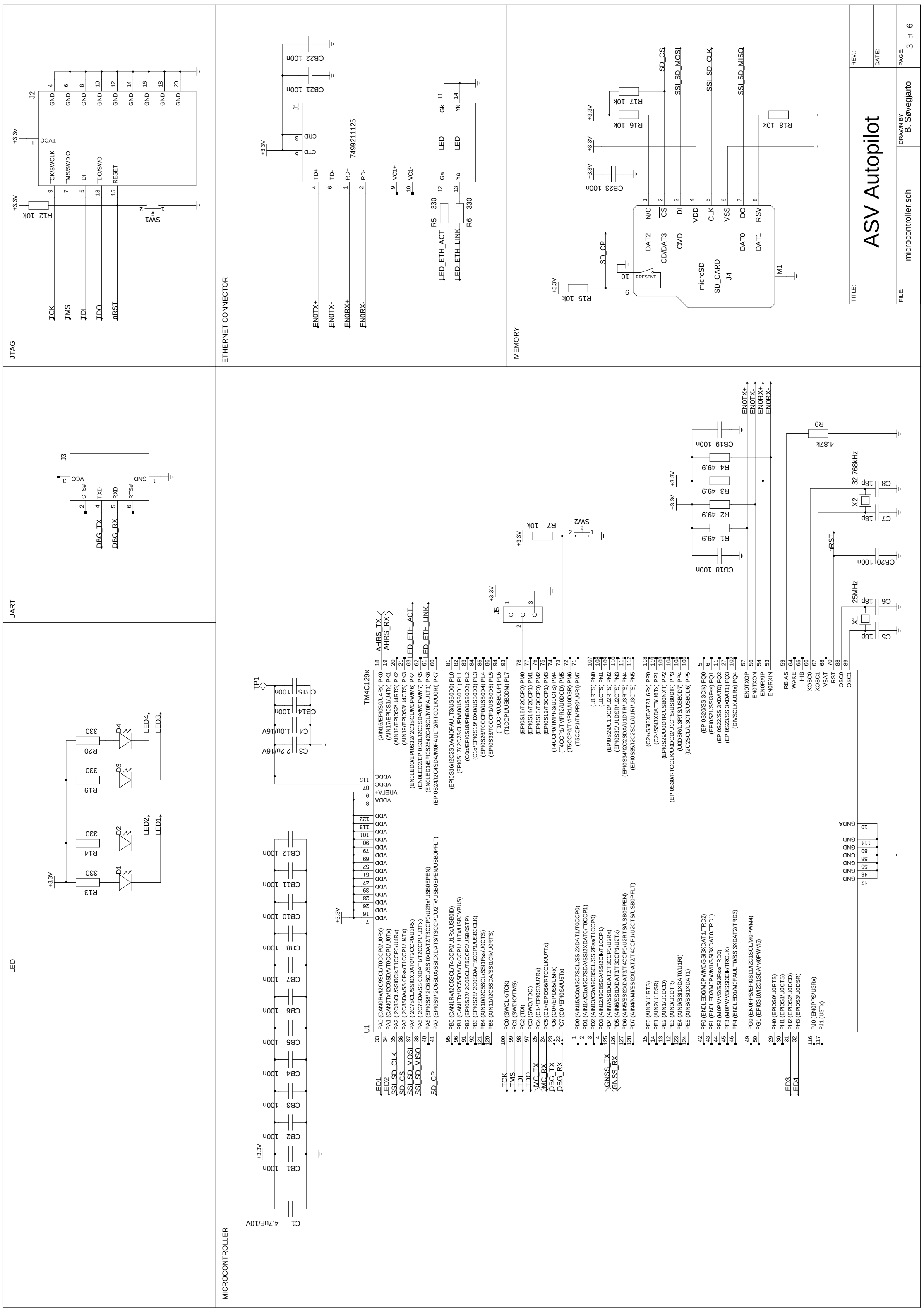
```
$( document ).ready( function() {

    var ready_for_new_wp = false;
    var draw_wanted = false;
    var got_message = false;

    var asv_planned_route = [
        [59.95990811, 10.79299452],
        [59.95978586, 10.79244960],
        [59.95966608, 10.79192318],
        [59.95951144, 10.79206993]
    ];

    var heading_canvas = $('div#heading␣canvas')[0];

    if (heading_canvas.getContext('2d')) {
        heading_ctx = heading_canvas.getContext('2d');
    } else {
        alert("Canvas␣not␣supported!");
    }

    var checkpoint_radius = 4;

    var asv_nr_of_msg = 0;

    /* Connect to socket */
    var socket = io.connect("/");

    socket.on('message', function(data) {

        //console.log(data);

        if (data.substring(0,6) == "STATUS") {

            $('div#status_bar').text(data);

            var data_array = data.split(",");

            asv_date = data_array[1];
            asv_time = data_array[2];
            asv_latitude = data_array[3];
            asv_longitude = data_array[4];
            asv_gps_ok = data_array[5];
            asv_current_heading = data_array[6];
            asv_wanted_heading = data_array[7];
            asv_distance_from_wp = data_array[8];
            asv_nr_of_wp = data_array[9];
            asv_mag_accuracy = data_array[10];
            asv_run_motors = data_array[11];
            asv_first_waypoint_latitude = data_array[12];
            asv_first_waypoint_longitude = data_array[13];
            asv_next_waypoint_latitude = data_array[14];
            asv_next_waypoint_longitude = data_array[15];
            asv_port_motor_thrust = data_array[16];
            asv_stbd_motor_thrust = data_array[17];

            $("span#asv_nr_of_wp").text(asv_nr_of_wp + "␣WP");
            $("span#asv_port_motor_thrust").text(asv_port_motor_thrust);
            $("span#asv_stbd_motor_thrust").text(asv_stbd_motor_thrust);

            $("span#asv_motor_running").siblings('i').removeClass("fa-spin");
            if (asv_run_motors == "1") {
                $("span#asv_motor_running").siblings('i').addClass("fa-spin");
                $("span#asv_motor_running").text("Running")
            } else {
                $("span#asv_motor_running").text("Not␣running")
            }

            if (asv_gps_ok == "1") {
                $("span#asv_gps_ok").text("OK").css('color', 'green');
                $("span#asv_date").text("20" + asv_date.substring(4,6) + "-" + asv_date
                    .substring(2,4) + "-" + asv_date.substring(0,2));
                $("span#asv_time").text(parseInt(asv_time.substring(0,2))+2 + ":" +
                    asv_time.substring(2,4) + ":" + asv_time.substring(4,6));
            } else {
                $("span#asv_gps_ok").text("Not␣OK").css('color', 'red');
                $("span#asv_date").text("N/A");
                $("span#asv_time").text("N/A");
            }

            if (parseInt(asv_mag_accuracy) >= 3) {
                $("span#asv_compass_ok").text("OK").css('color', 'green');
            } else {
                $("span#asv_compass_ok").text("Not␣OK").css('color', 'red');
            }

            if (asv_gps_ok = "1" && parseInt(asv_first_waypoint_latitude) != 0 &&
                parseInt(asv_first_waypoint_longitude) != 0) {
                $("span#asv_distance_from_wp").text(asv_distance_from_wp + "␣m␣to␣WP");
                draw_wanted = true;
            } else {
                $("span#asv_distance_from_wp").text("N/A");
                draw_wanted = false;
            }

            $("span#asv_nr_of_msg").text(++asv_nr_of_msg);
            $("span#asv_nr_of_msg").siblings('i').addClass("fa-spin");

            if (asv_nr_of_msg >= 99) {
                asv_nr_of_msg = 0
            }
            got_message = true;

            draw_indicator(heading_ctx, asv_wanted_heading, asv_current_heading,
                draw_wanted);

            asv_marker.options.angle = asv_current_heading; //direction * (180 / Math.
                PI);
            asv_marker.update();

            if (asv_latitude != 0 && asv_longitude != 0) {
                var asv_position = new L.LatLng(asv_latitude, asv_longitude);
                asv_marker.setLatLng(asv_position);
                asv_history.addLatLng(asv_position);
            }

            first_waypoint.setLatLng(new L.LatLng(asv_first_waypoint_latitude,
                asv_first_waypoint_longitude));
            next_waypoint.setLatLng(new L.LatLng(asv_next_waypoint_latitude, asv_next_waypoint_longitude));

            if (parseInt(asv_nr_of_wp) < 2 && ready_for_new_wp == true) {
                //planned_checkpoints.getLayers()[0].spliceLatLngs(0,1)[0];
                //var last_cleared_checkpoint = planned_checkpoints.getLayers()[0].getLatLngs()[0];
                //console.log("checkpoint at latitude " + last_cleared_checkpoint.lat + " and longitude " + last_cleared_checkpoint.lng + " cleared");
                //cleared_checkpoints.addLatLng(last_cleared_checkpoint);
                //next_checkpoint.setLatLng(last_cleared_checkpoint);
                //socket.emit('message', 'NWP,'+ last_cleared_checkpoint.lat + "," + last_cleared_checkpoint.lon);
                var next = asv_planned_route.shift();
                socket.emit('message', 'NWP,'+ next[0] + "," + next[1]);
                ready_for_new_wp = false;
            }
        } else if (data.substring(0,6) == "ACKRWP") {
                ready_for_new_wp = true;
        }
    });

    socket.emit('message', 'ASV␣Control␣Panel␣started');

    /* Init map */
    var map = L.map('map', {zoomControl: false}).setView([59.95974031, 10.79310980], 19);
    L.tileLayer('img/tiles/{z}/{x}/{y}.png', {
            zoom: 21,
            maxZoom: 21,
            detectRetina: true
            }).addTo(map);

    map.dragging.disable();
    map.touchZoom.disable();
    map.doubleClickZoom.disable();
    map.scrollWheelZoom.disable();
    map.boxZoom.disable();
    map.keyboard.disable();

    var asv_marker = L.rotatedMarker(new L.LatLng(59.959707, 10.792272), {
        icon: L.icon({
            iconUrl: 'img/asv_marker.png',
            iconRetinaUrl: 'img/asv_marker@2x.png',
            iconSize: [26, 50],
            iconAnchor: [13, 25],
        })
    });

    asv_marker.addTo(map);

    /* Init buttons */

    L.easyButton('fa-send', function() {
        ready_for_new_wp = true;
    }, "Send␣WP", map);

    L.easyButton('fa-trash', function() {
        socket.emit('message', 'DEL');
    }, "Delete␣WP", map);

    L.easyButton('fa-play', function() {
        socket.emit('message', 'RUN');
    }, "Start␣the␣ASV", map);

    L.easyButton('fa-stop', function() {
        socket.emit('message', 'HLT');
    }, "Stop␣the␣ASV", map);

    /* Init planned checkpoints */
    var planned_checkpoints = new L.FeatureGroup();
    map.addLayer(planned_checkpoints);

    var draw_control = new L.Control.Draw({
        draw: {
            circle: false,
            rectangle: false,
            polygon: false,
            marker: false
        },
        edit: {
            edit: true,
            featureGroup: planned_checkpoints
        }
    });
    map.addControl(draw_control);

    map.on('draw:created', function(e) {
        console.log(e.layer.getLatLngs());
        planned_checkpoints.addLayer(e.layer);
    });

    map.on('draw:edited', function(e) {
        e.layers.eachLayer(function (layer) {
            console.log(layer.getLatLngs());
        });
    });

    /* Init asv history */
    var asv_history_points = [ [59.95974031, 10.79310980] ];

    var asv_history_options = {
        smoothFactor: 1,
        color: '#000'
    };

    var asv_history = L.polyline(asv_history_points, asv_history_options).addTo(map);

    /* Init cleared checkpoints */
    var cleared_checkpoints_points = [];

    var cleared_checkpoints_option = {
        smoothFactor: 0,
        color: '#00f'
    };

    var cleared_checkpoints = L.polyline(cleared_checkpoints_points,
        cleared_checkpoints_option).addTo(map);

    /* Init future waypoint markers */
    var waypoint_options = {
        color: '#f00',
        opacity: 1,
        weight: 10,
        fillColor: '#f00',
        fillOpacity: 0.5
    };
    var first_waypoint = L.circle([0, 0], checkpoint_radius, waypoint_options).addTo(
        map);
    var next_waypoint = L.circle([0, 0], checkpoint_radius/2, waypoint_options).addTo(
        map);


    var circle_options = {
        color: '#ff0',
        opacity: 1,
        weight: 3
    };

    L.circle([59.95974031, 10.79310980], 1, circle_options).addTo(map);
    L.circle([59.95990811, 10.79299452], 1, circle_options).addTo(map);
    L.circle([59.95978586, 10.79244960], 1, circle_options).addTo(map);
    L.circle([59.95966608, 10.79192318], 1, circle_options).addTo(map);
    L.circle([59.95951144, 10.79206993], 1, circle_options).addTo(map);

    window.setInterval(function() {
        if (!map.getBounds().contains(asv_marker.getLatLng())) {
            //map.panTo(asv_marker.getLatLng());
        }

        if (got_message == false) {
            $("span#asv_nr_of_msg").siblings('i').removeClass("fa-spin");
        }
        got_message = false;

    }, 1000);

    function clear_canvas(ctx) {
        ctx.clearRect(0, 0, 200, 200);
    }

    function draw(degrees) {

        clear_canvas();

        // Draw the compass onto the canvas
        ctx.drawImage(compass, 0, 0);

        // Save the current drawing state
        ctx.save();

        // Now move across and down half the
        ctx.translate(100, 100);

        // Rotate around this point
        ctx.rotate(degrees * (Math.PI / 180));

        // Draw the image back and up
        ctx.drawImage(needle, -100, -100);

        // Restore the previous drawing state
        ctx.restore();
    }

    function draw_indicator(ctx, degrees_wanted, degrees_current, draw_wanted) {

        ctx.clearRect(0, 0, 200, 200);

        var center_x = 100;
        var center_y = 100;

        // Draw outer circle
        ctx.beginPath();
        ctx.arc(center_x, center_y, 100, 0, 2*Math.PI, false);
        ctx.lineWidth = 1;
        ctx.strokeStyle = '#000';
        ctx.fillStyle = "rgba(0,0,0,0.1)";
        ctx.fill();

        // Draw N, S, E, W
        ctx.save();
        ctx.translate(center_x, center_y);
        ctx.fillStyle = "black";
        ctx.font = "1.3em␣open_sansregular";
        ctx.textBaseline = "middle";
        ctx.textAlign = "center";
        ctx.save();
        ctx.translate(0, -60);
        ctx.fillText("N", 0, 0);
        ctx.restore();
        ctx.save();
        ctx.translate(60, 0);
        ctx.fillText("E", 0, 0);
        ctx.restore();
        ctx.save();
        ctx.translate(0, 60);
        ctx.fillText("S", 0, 0);
        ctx.restore();
        ctx.save();
        ctx.translate(-60, 0);
        ctx.fillText("W", 0, 0);
        ctx.restore();
        ctx.restore();

        if (draw_wanted == true) {
            // Draw wanted heading
            ctx.lineWidth = 2;
            ctx.setLineDash([2]);
            ctx.strokeStyle = '#f00';
            draw_needle(90, degrees_wanted);
            ctx.setLineDash([]);
        }

        // Draw needle mounting
        ctx.beginPath();
        ctx.arc(center_x, center_y, 5, 0, 2 * Math.PI, false);
        ctx.fillStyle = '#00F';
        ctx.fill();
        ctx.closePath();

        draw_tics();

        // Draw current heading
        ctx.lineWidth = 2;
        ctx.strokeStyle = '#00F';
        draw_needle(90, degrees_current);

        function draw_tics() {
            ctx.strokeStyle = "black";
            ctx.lineWidth = 1;

            // 360/72 = 5 degrees
            var nr_of_tics = 72;

            for (var i = 0; i < nr_of_tics; i++) {
                ctx.save();
                ctx.beginPath();
                ctx.translate(center_x, center_y);
                var angle = (i * (360/nr_of_tics)) * Math.PI/180;
                ctx.rotate(angle);
                ctx.translate(0, -180/2);

                ctx.moveTo(0, 0);
                ctx.lineTo(0, 10);
                ctx.stroke();
                ctx.closePath();
                ctx.restore();
            }
        }

        function draw_needle(length, angle) {
                ctx.save();
                ctx.beginPath();
                ctx.translate(center_x, center_y);

                // Correct for top left origin
                ctx.rotate(-180 * Math.PI/180);

                ctx.rotate(angle * Math.PI/180);
                ctx.moveTo(0, 0);
                ctx.lineTo(0, length);
                ctx.stroke();
                ctx.closePath();
                ctx.restore();
        }

    }
});
```

Appendix O

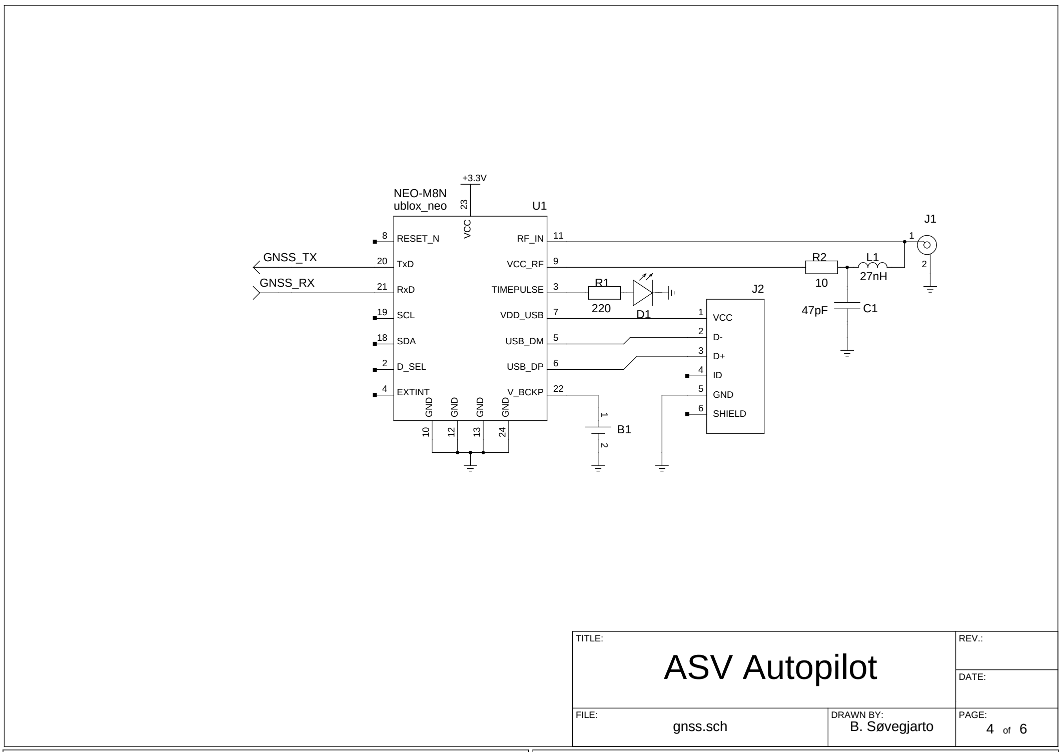
Autopilot schematics



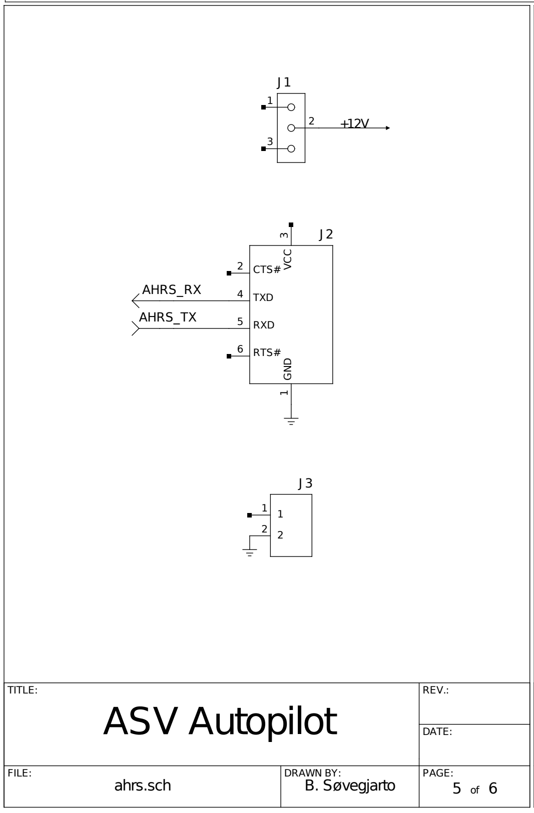

VIN
D1
GF1K-E3/67A
F1
TP8
4.7uF/100V
C8
TEN 3-2410WIN
U2
22
23
+Vin
+Vin
2
3
-Vin
-Vin
14
+Vout
-Vout
16
PGND
PGND
10uF/50V
C11
100nF/50V
CB3
220
R5
D6
TP4
VCC
TP7
TITLE:
ASV Autopilot
REV.:
DATE:
FILE:
power_supply_unit.sch
DRAWN BY:
B. Søvegjarto
PAGE:
2 of 6



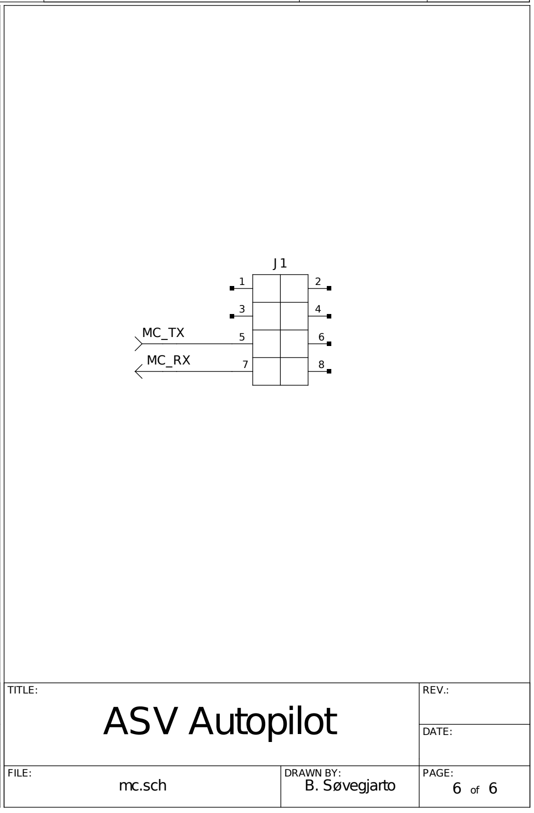
JTAG
ETHERNET CONNECTOR
MEMORY
UART
LED
MICROCONTROLLER
TM4C129x
U1
J1
J2
J3
J4
J5
SD_CARD
microSD
TITLE:
ASV Autopilot
DRAWN BY:
B. Søvegjarto
FILE:
microcontroller.sch
PAGE:
3 of 6



+3.3V
NEO-M8N
ublox_neo
U1
VCC
RESET_N
TxD
RxD
SCL
SDA
D_SEL
EXTINT
RF_IN
VCC_RF
TIMEPULSE
VDD_USB
USB_DM
USB_DP
V_BCKP
GND
GNSS_TX
GNSS_RX
R1
220
D1
R2
10
L1
27nH
47pF
C1
J1
J2
VCC
D-
D+
ID
GND
SHIELD
B1
TITLE:
ASV Autopilot
REV.:
DATE:
FILE:
gnss.sch
DRAWN BY:
B. Søvegjarto
PAGE:
4 of 6



J1
+12V
J2
VCC
CTS#
TXD
RXD
RTS#
GND
AHRS_RX
AHRS_TX
J3
TITLE:
ASV Autopilot
REV.:
DATE:
FILE:
ahrs.sch
DRAWN BY:
B. Søvegjarto
PAGE:
5 of 6



J1
MC_TX
MC_RX
TITLE:
ASV Autopilot
REV.:
DATE:
FILE:
mc.sch
DRAWN BY:
B. Søvegjarto
PAGE:
6 of 6

Bibliography

[1] European Parliament, Council of the European Union, "Directive 2000/60/EC of the European Parliament and of the Council of 23 October 2000 establishing a framework for Community action in the field of water policy." `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32000L0060`, Accessed: 2014-10-14.

[2] FFI, "Hugin Autonomous Underwater vehichle (AUV)." `http://www.ffi.no/en/research-projects/Hugin/Sider/default.aspx`, Accessed: 2014-10-14.

[3] M. Neal, "A Hardware Proof of Concept of a Sailing Robot for Ocean Observation," *Oceanic Engineering, IEEE Journal of*, vol. 31, pp. 462–469, April 2006.

[4] P. Liu, W. Farias, S. Gibson, and D. Ross, "Remote control of a robotic boat via the Internet," in *Information Acquisition, 2005 IEEE International Conference on*, pp. 6 pp.–, June 2005.

[5] "er9x – Custom firmware for the Eurgle/FlySky/Imax/Turnigy 9x r/c Transmitter." `http://code.google.com/p/er9x/`, Accessed: 2014-10-28.

[6] "th9x – Alternative Firmware for 9-ch FlySky RC-Controler." `http://code.google.com/p/th9x/`, Accessed: 2015-01-07.

[7] A. Tantos, "H-Bridges – the Basics," `http://modularcircuits.tantosonline.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/?format=pdf`, Accessed: 2015-03-06.

[8] Y. Xiong, S. Sun, H. Jia, P. Shea, and Z. Shen, "New Physical Insights on Power MOSFET Switching Losses," *Power Electronics, IEEE Transactions on*, vol. 24, pp. 525–531, Feb 2009.

[9] "IPC-2221A Generic Standard on Printed Board Design," May 2003.

[10] D. C. Giancoli, *Physics for Scientists & Engineers with Modern Physics (4th Edition)*. Addison-Wesley, 2009.

[11] B. Blick, "Servo pulse to PWM converter." `http://www.bobblick.com/techref/projects/sv2pwm/sv2pwm.html`, Accessed: 2014-10-28.

[12] S. Lloyd, M. McFadden, D. Jennings, R. L. Doerr, and C. Baron, "Supporting documentation for the OSMC project." `http://www.robotpower.com/downloads/OSMC_project_documentation_V4_25.pdf`, Accessed: 2015-01-11.

[13] "LFPAK – The Toughest Power-SO8." `http://www.nxp.com/documents/leaflet/939775016838_LR.pdf`, Accessed: 2015-04-02.

[14] "LFPAK MOSFET thermal design guide (AN10874)." http://www.nxp.com/documents/application_note/AN10874.pdf, Accessed: 2015-04-02.

[15] J. K. Jensen, "Attitude Estimation for Motion Stabilization in Sonar Systems," Master's thesis, University of Oslo, 2013. http://urn.nb.no/URN:NBN:no-38644.

[16] D. Titterton and J. Weston, *Strapdown Inertial Navigation Technology (IEE Radar, Sonar, Navigation and Avionics Series).* The Institution of Engineering and Technology, 2005.

[17] "MPU-9150 Product Specification." http://www.invensense.com/mems/gyro/documents/PS-MPU-9150A-00v4_3.pdf, Accessed: 2015-05-03.

[18] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, pp. 15–16, 2006.

[19] "Atmel AVR1916: USB DFU Boot Loader for XMEGA." http://www.atmel.com/images/doc8429.pdf, Accessed: 2015-02-06.

[20] "USB Frequently Asked Questions." http://www.usb.org/developers/usbfaq, Accessed: 2015-04-07.

[21] "The Evolution of Copper Cabling Systems from Cat5 to Cat5e to Cat6." http://www.gocsc.com/UserFiles/File/Panduit/Panduit098765.pdf, Accessed: 2015-04-19.

[22] R. Koprowski, Z. Wrobel, A. Kleszcz, S. Wilczynski, A. Woznica, B. Lozowski, M. Pilarczyk, J. Karczewski, and P. Migula, "Mobile sailing robot for automatic estimation of fish density and monitoring water quality," *BioMedical Engineering OnLine*, vol. 12, no. 1, p. 60, 2013.

[23] C. A. Goudey, T. Consi, J. Manley, M. Graham, B. Donovan, and L. Kiley, "A robotic boat for autonomous fish tracking," *Marine Technology Society*, vol. 32, no. 1, pp. 47–53, 1998.

[24] M. Chaumet-Lagrange, "Nautical apparatus to conduct reconnaissance missions of a site, particularly bathymetric surveys," Nov. 18 1997. US Patent 5,689,475.

[25] A. Dhariwal, A. A. de Menezes Pereira, C. Oberg, B. Stauffer, S. Moorthi, D. A. Caron, G. Sukhatme, *et al.*, "NAMOS: Networked aquatic microbial observing system," *Center for Embedded Network Sensing*, 2006.

[26] B. G. Liptak, *Instrument Engineers' Handbook,Third Edition: Process Control.* CRC Press, 1995.

[27] S. Bennett, "Nicholas minorsky and the automatic steering of ships," *Control Systems Magazine, IEEE*, vol. 4, pp. 10–15, November 1984.

[28] "Atmel AVR221: Discrete PID controller." http://www.atmel.com/images/doc2558.pdf, Accessed: 2015-04-19.

[29] "RFC6455 – The WebSocket Protocol." http://tools.ietf.org/html/rfc6455, Accessed: 2015-04-06.

[30] "Radionavigasjon (DGPS)." http://www.kystverket.no/Maritime-tjenester/Meldings--og-informasjonstjenester/Radionavigasjon-DGPS/, Accessed: 2015-05-05.

[31] "CPOS." http://kartverket.no/Posisjonstjenester/CPOS/, Accessed: 2015-05-05.

[32] T. Takasu and A. Yasuda, “Development of the low-cost RTK-GPS receiver with an open source program package RTKLIB,”

[33] F. van Diggelen, *A-GPS: Assisted GPS, GNSS, and SBAS.* Artech House, 2009.

[34] R. W. Sinnott, “Virtues of the Haversine,” *Sky and Telescope*, vol. 68, p. 158, Dec. 1984.

[35] Ed Williams, “Aviation Formulary V1.46.” `http://williams.best.vwh.net/avform.htm#Crs`, Accessed: 2015-05-05.

[36] T. Takasu and A. Yasuda, “Evaluation of RTK-GPS Performance with Low-cost Single-frequency GPS Receivers,”

[37] “Standard Group MAC Addresses: A Tutorial Guide.” `http://standards.ieee.org/develop/regauth/tut/macgrp.pdf`, Accessed: 2015-04-12.

[38] “RFC2365 – Administratively Scoped IP Multicast.” `http://tools.ietf.org/html/rfc2365`, Accessed: 2015-04-12.

[39] “Magnetic Declination Estimated Value.” `http://www.ngdc.noaa.gov/geomag-web/`, Accessed: 2015-05-03.

[40] “Magnetic declination calculator.” `http://geomag.nrcan.gc.ca/calc/mdcal-eng.php`, Accessed: 2015-05-03.

[41] B. Gati, “Open source autopilot for academic research - The Paparazzi system,” in *American Control Conference (ACC), 2013*, pp. 1478–1481, June 2013.