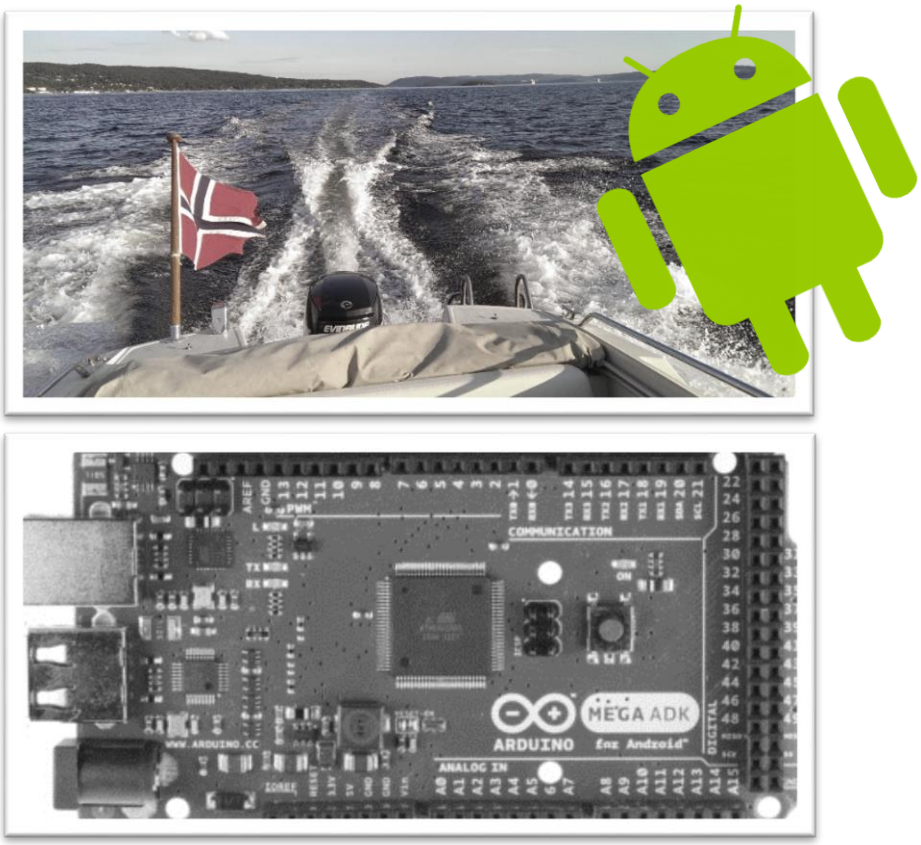


PROJECT REPORT AUXILIARY CONTROL SYSTEM LEISURE BOAT

2013

Emil Niclas Hofseth | Electronics project | November 18, 2013

ABSTRACT

The following is a project report for my project in the subject «Elektroprosjekt LV388E» (Electronics project) in the 5th semester at Oslo and Akershus University College of Applied Sciences. I have worked on this project throughout the semester, from planning through execution, to the writing of this report, independently by myself.

The goal for the project was to produce a platform for an auxiliary control system for leisure boats, giving the crew a possibility to read out different statuses for the boat, as well as controlling some of the systems onboard from an Android-based application. This in combination with an autonomous relay, securing enough battery for the engine, independent of the power drainage produced by onboard auxiliary systems.

Since this is a project near the end of the degree, I wanted to base it on the pillars of knowledge I have built up so far, including parts of programming, electronics, design, production, as well as project planning. All in all this has been a demanding project, concerning both level of learning and time consumption. However, put in perspective of the broadness of the task, I feel satisfied with the finished product, and the foundation it is for further development.

CONTENTS

Abstract 2
Figures and tables 4
Introduction 5
Planning 7
Contretization of goal 7
Battery 7
Android application 8
Sensors 8
Preliminary goal 8
Hardware 9
Design 9
Sensing charge 9
Sensors and microcontroller 10
Production 12
Software 18
Learning 18
History 18
Android 18
Eclipse – IDE 19
Production 19
Factory Acceptance test - FAT 21
Minutes of Meeting 22
Meeting number 1 - 24th september 22
Meeting number 2 – 7th october 22
Meeting number 3 – 4th november 22
Conclusion and potential 23
Shortcommings 23
Potential 23
Conclusion 23
Appendix 1: Gantt-chart 24
Appendix 2: Project Presentation (In Norwegian) 25
Appendix 3: Android source code 27
Appendix 4: Arduino source code 33
Appendix 5: Factory acceptance test for aux. control system 37

Appendix 6: Status meeting nr. 1 attachment 39

FIGURES AND TABLES

Figure 1 - Building blocks 5
Figure 2 – Preliminary sketch of system 8
Figure 3 - Voltage comparator 9
Figure 4 - Relay 10
Figure 5 - EM-406 GPS-module 10
Figure 6 - FS-3400AH Fuel flow sensor 10
Figure 7 - Arduino Android Developer Kit 11
Figure 8 - Prototype on breadboard. 12
Figure 9 - Etch masking for PCB production 13
Figure 10 - Picture of PCB pre soldering 13
Figure 11 - Sensors and microcontroller 14
Figure 12 - Arduino IDE 14
Figure 13 – Generic Arduino lifecycle 15
Figure 14 - Arduino code flowchart 15
Figure 15 - message protocol 16
Figure 16- Anrdoid architecture(wikipedia) 18
Figure 17 - Android flowchart 20
Figure 18 - Gantt chart 24

INTRODUCTION

As the description for this subject states, the students should be divided in groups of 2-4 students for a suitable project. Since I am currently working 20% next to school and taking subjects from both the second and the 3rd year, my schedule is best suited for working as an individual, and that has been the solution for this subject as well. I have worked independently with this project from the start of the semester, even though the though process started immediately after we chose this autumns courses, in late June earlier this year. Almost every second week I have had a status meeting with my supervisor, Veslemøy Tyssø, updating her on the progress of the project.

My goal for this project was to build upon the previous subjects, ranging from programming, to electronics and project planning, see Figure 1 - Building blocks. Even though I might have not been able to go as deep into the matter as I might have had, if I were to focus on just one of the pillars – I think that an important part of being an engineer, is to be able to have an oversight and an ability to implement different divisions of engineering into one product.

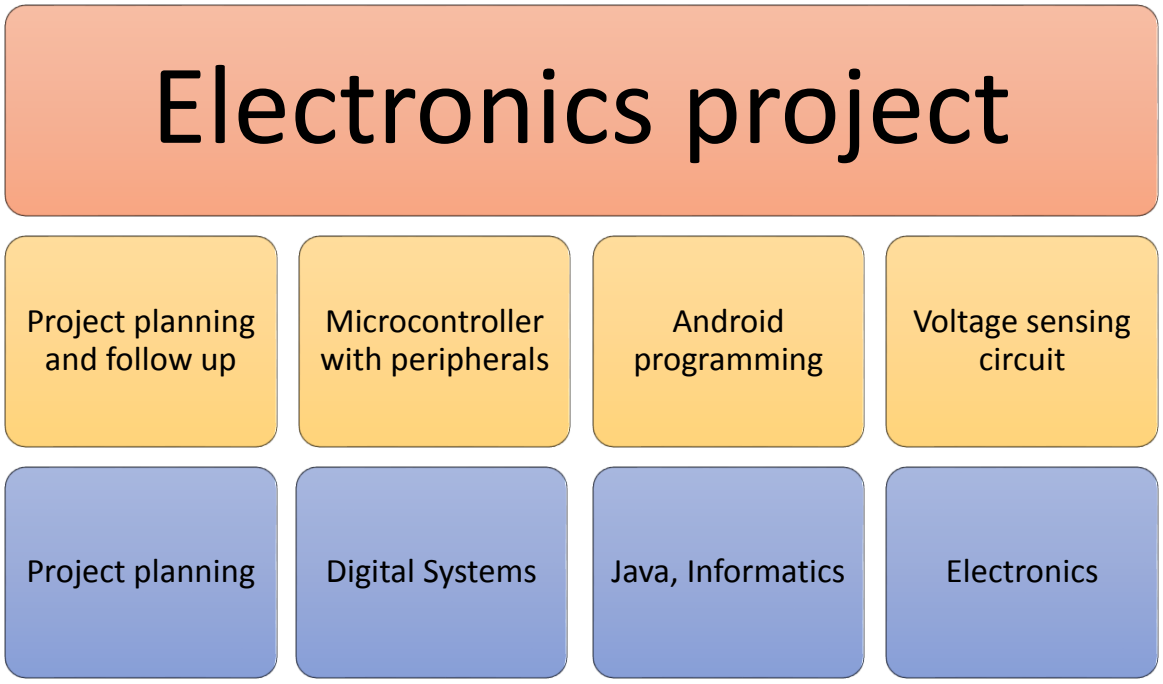


FIGURE 1 - BUILDING BLOCKS

As previously stated, I selected this autumn's course in late June. That gave me the summer to materialize a goal for the project. Spending quite a lot of time in my boat, I discovered the need for an additional battery. The one I already had is primarily for the starting of the outboard motor and I could not risk not being able to start the engine, due to discharging of the battery on auxiliary items, such as charging mobile phones, laptops or a refrigerated box for food. To be able to charge this auxiliary battery from the same generator in the engine, and without discharging the start battery when the engine is not running, a switch is needed to disconnect these batteries while at anchor. Furthermore, as my boat is very old, predating modern GPS and even basic instruments, I wanted some sort of display, showing me current speed, charging voltages and maybe even fuel consumption. Boats with V-hulls are very sensitive to weight for planing, which again reflects on the

fuel usage and there again economical cruising speed. As well as at top speed, it is difficult to tell when the fuel consumption vs. additionally gained speed is inefficient.

PLANNING

I had to go from being onboard, thinking out potential projects and solutions, to delivering a written suggestion to my supervisor, Mrs. Tyssø.
Having such a diverse project in head of me, I needed to divide the tasks into several stages. To exercise the project planning I chose Microsoft Project as my tool for planning and follow through of the project. This enabled me to break down the project in different tasks, stages and resources. The plan was to use this tool throughout the project, keeping the timetable and resource-usage dynamic, but an unforeseen computer crash midway through the project, without sufficient backup, stopped this. Still, the baseline Gantt-chart is still usable. Due to the loss of the original files, the attached chart (Appendix 1: Gantt-chart) is in Norwegian. The letter with a presentation of the project is attached (Appendix 2: Project Presentation), even though it also is in Norwegian, for documenting purposes.

The first step of planning was to separate hardware and software. From there, the different stages of a new product was included – from design, through production and test, to the finished product. I had to estimate the usage of resources, both man-hours and production parts. This is an art based on experience, and we will see later in the report that the estimates was diverging in some cases. The majority of hours was allocated to the software division of the project, because of the new and unknown Android programming.

According to the Gantt-chart, the project was divided into the following steps, for both the software and the hardware division:

- Design
- Production
- Test

In addition, the software division had an extra step called learning – including the reading and basic training to get up to speed on the Android programming.

At last there is set a schedule for the FAT(Factory Acceptance Test), normally the last step before the product leaves the factory and is taken over by the customer (for later HAT and SAT – Harbour/Sea Acceptance Test).

CONTRETIZATION OF GOAL

BATTERY

As previously stated, the batteries needs to be separated when the engine is not running, and the generator therefore is not delivering power. In its most basic form, this functionality could be produced by a simple manual switch. Taking it one step further a relay could do this, controlled from a remote switch. Even a further step could be to substitute the switch with a GUI (Graphical User Interface) in an Android application, controlling the relay, connecting and disconnecting the batteries when needed. The latter is a wanted functionality. However, the problem arises if the Android unit is turned off, either on purpose or by accident. To solve this, and to implement the pillar of electronics to the project, I wanted to create a form of automatic relay, functioning autonomously without crew interference. If wanted, the relay should be override-able, in case of emergency or other uses. This sub goal will cover the basic electronics, and put to play knowledge from the subject of “Electronics”.

Android application

The GUI should be able to view at least vessel speed, fuel consumption and remaining fuel, as well as battery voltage for the respective batteries. It is also a wanted functionality to be able to override the automatic relay, connecting the batteries on demand. Building an Android application will be a level up from the previous completed course “Informatics” where Java is introduced.

Sensors

To be able to collect data such as the fuel consumption and speed, various sensors are needed. To collect the sensor information and process them before sending them to the GUI, some sort of programmable microcontroller could be used as an interface. This will also let me further develop the skills learned in the courses of “Digital Systems”.

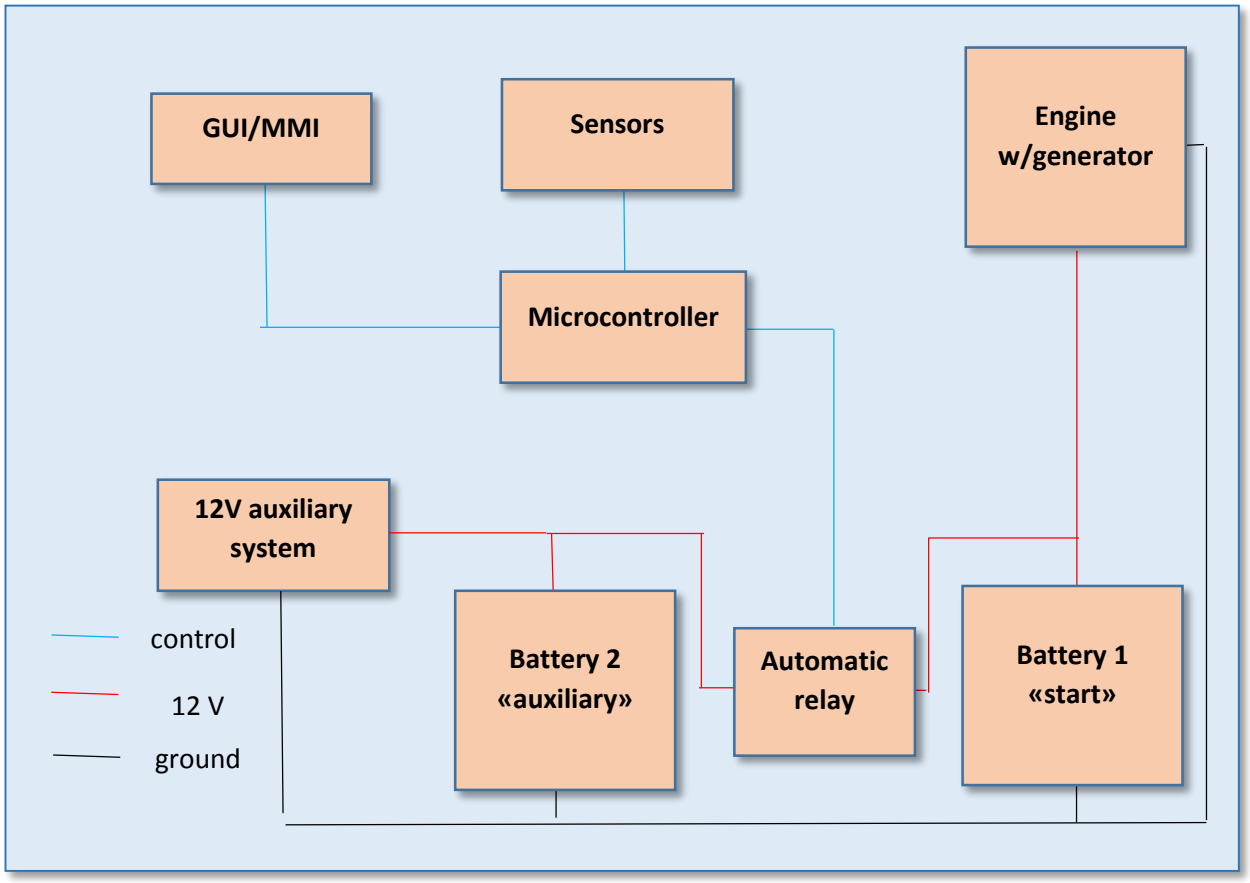


Figure 2 – Preliminary sketch of system

Preliminary goal

After concretizing the goals, the project plan could be seen as in Figure 2. This was the baseline for further development.

HARDWARE

For this project, the hardware division is a collective term for the part of the project including the relay, low-level microcontroller programming and sensor management.

DESIGN

SENSING CHARGE

The idea is that the starter engine gets it's current from the starting battery. This will drop the charge level on this battery, whom will soon be recharged in a matter of 15-20 minutes, regarding of the size of starter engine and the time it takes to start the engine.

I quickly discovered that a unit for automatically connecting and disconnecting batteries based on the level of charge, already existed. The units are called "voltage sensitive relays" and cost approximately 900 NOK. I found no blueprints of these systems online, but this tipped me of that sensing the voltage might be a good place to start. Alternative ways of measuring the charge level, could in example be to measure the charging current and finding the suitable level for when to connect the other battery. The disadvantage in this is the foreseeing of currents regards to different charging levels and combinations of those. This alternative was not investigated further, due to the discovery of the units sensing voltage.

The next step would be to find a stable way of detecting the voltage. This brought use of the knowledge of operational amplifiers (OPAMP), acquired in the "Electronics"-course.

VOLTAGE COMPARATOR

A voltage comparator is an OPAMP configured in such a way that it can produce an outgoing voltage level, based on the difference between the input voltages. In the case of Figure 3, the V_{out} will rise when V_1 surpasses V_2. If we introduce a pull-up resistor on the output, we have the voltage comparator producing a HIGH signal when $V_1 > V_2$.

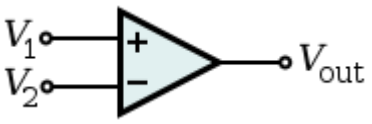


FIGURE 3 - VOLTAGE COMPARATOR

In our case the V_1 should be the voltage level of the battery, and the V_2 should be a reference voltage hereby named V_{ref}. V_{out} should then be a proficient signal to trigger a relay.

Taking in to consideration that the entire system is being run of the same power source, namely the battery and the generator, finding a suitable reference could be a challenge. For practical and economic reasons, I landed on using a Zener diode for my project. The diodes breakdown voltage is a good enough reference, and its temperature stability is fair enough for the scope of this project. If this assumption proves to be incorrect in the long term, an option is to replace the Zener with a band gap voltage reference. This is a pre-produced package, which delivers a temperature independent fixed voltage, regardless of power supply variations and load

Since a Zener diode is already available to the project, it is used in combination with a voltage divider, which divides the input voltage from the battery to a level that easily can be compared with the Zeners breakdown voltage and together with a pull-up resistor produce a signal to trigger a relay.

RELAY

The boats electrical system is running at 12 V, similar to automobiles. Due to the great currents involved when, both starting the engine and charging the batteries I ensured enough margin when buying a relay that manages both 12 V and up to 100 A. A valid datasheet for this item is yet to be found, but bears the serial number “160477z” from “HC Cargo”, as in Figure 4. It is a Single Pole Simple Throw , Relay(SPST), or better known as a switch. When current is introduced to the ports 85/86, a coil makes a magnetic field, which again makes the 30/87 ports open for a greater current – in this case the connection between the positive terminals of the two batteries. Price: Aprox. 200 NOK.

FIGURE 4 - RELAY

SENSORS AND MICROCONTROLLER

To get inputs on speed, fuel consumption and voltage level of the batteries the following sensors and microcontroller were chosen for the project.

GPS

EM-406 GPS-module. This module is based on the SiRF StarIII chipset. It includes onboard voltage regulation, antenna and battery backed RAM.

- 20 Channels
- Sensitivity: -159dBm
- Accuracy: 10m
- Power: 70mA at ~5V
- Output: NMEA 0183
- Dimensions: 30x30x10.5mm
- Weight: 16g
- Price: 244 NOK

FIGURE 5 - EM-406 GPS-MODULE

FUEL FLOW SENSOR

Savant FS-3400AH Diesel Flow Sensor. A simple unit based on a impeller and a magnet, producing an output signal that varies with fuel flow. Finding a unit that was resistant to fuel was paramount – the fuel would damage a regular water flow sensor.

- Wide supply voltage range: 2.4-26V
- Power: 2.8 A
- Accuracy: +/- 10%
- Price: 80 NOK

Flow rate	Liter / pulse
2.0	0.00038
4.0	0.00050
12.0	0.00050
30.0	0.00050

FIGURE 6 - FS-3400AH FUEL FLOW SENSOR

MICROCONTROLLER

Arduino ADK, based on the ATmega 2560, 8-bit microcontroller with 256KB flash. This unit was chosen, since it has an onboard USB-port for serial communication with Android units. It also offers a great selection of I/O ports, for further development of the project.

- 54 digital input/output
- 16 analog input
- 4 UART
- USB connection
- Price: 400 NOK

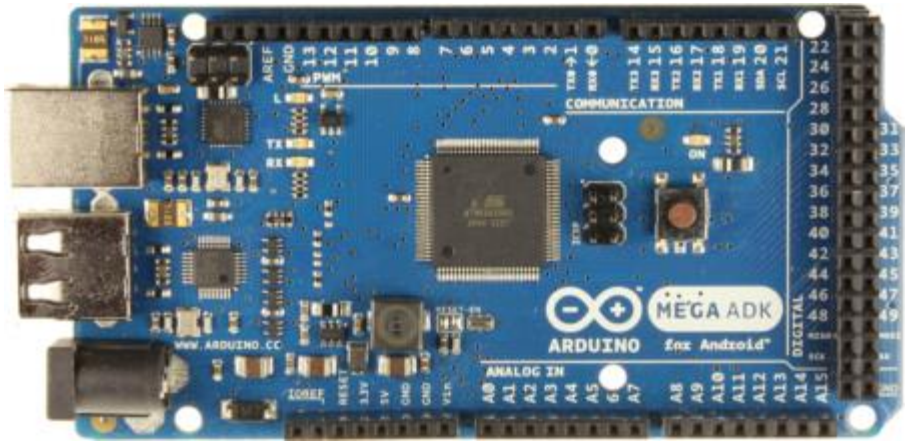


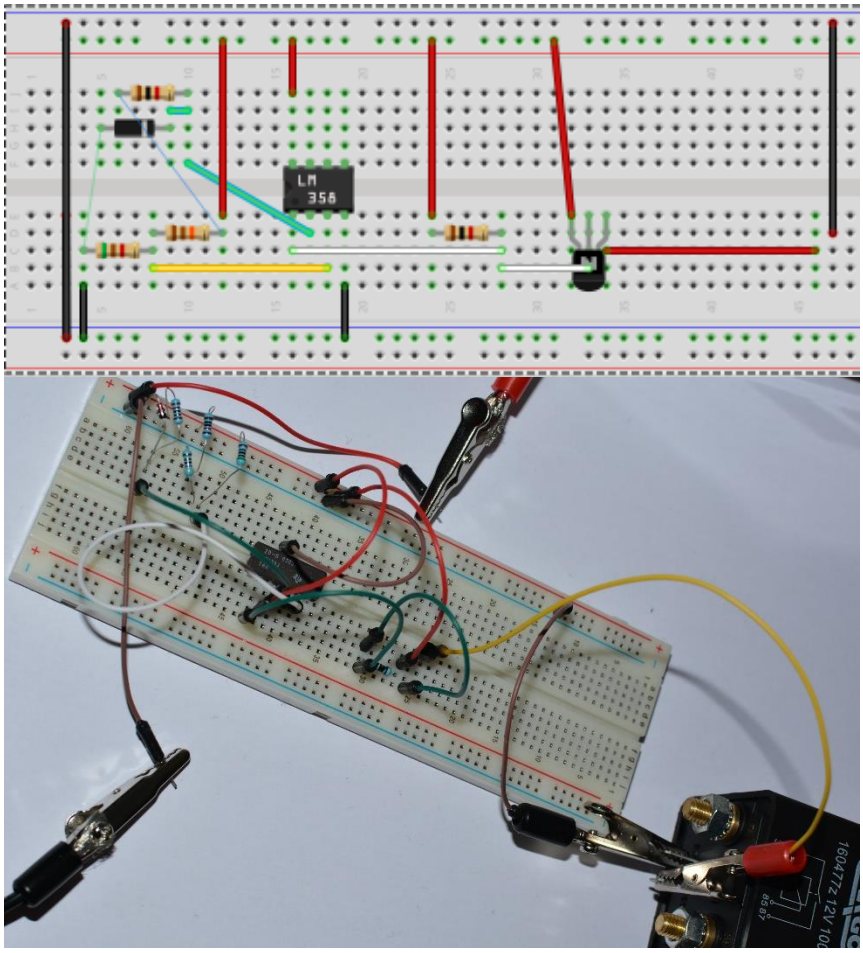
FIGURE 7 - ARDUINO ANDROID DEVELOPER KIT

PRODUCTION

PROTOTYPING

The next step was bringing together the voltage comparator and the relay. To get the correct values for the circuit, I referred to the producer of the battery, stating a voltage level of 13.5V when the battery is fully charged – during charging. When fully charged and not charging, the voltage level is about 12.7V. This results in dividing the 13.5V down to a comparable size to the Zener.

The first prototype was assembled on a breadboard. The following elements were chosen:



- Zener diode; 3.3V breakdown voltage.
- Voltage divider: R1 = 14.7kΩ and R2 = 5.1kΩ resulting in: $V_{out} = \frac{R_2}{R_1+R_2} \times V_{in} = 3.45V$

My acquired resistors have a low degree of accuracy, so 3.45V is good enough for further development.

It was quickly discovered that the relay was more current demanding than assumed. This was easily solved by placing a Bipolar Junction Transistor (BJT) in the circuit, controlled by the comparator and supplying the coil.

FIGURE 8 - PROTOTYPE ON BREADBOARD.

The test of the breadboard prototype was a success. The distinct sound of the coil in the relay clicking when the supply voltage simulating the start battery surpassing 13.5 V. To ensure validity, connection was measured between the 30/87 ports on the relay.

PRINTING

According to plan, I was to produce a printed circuit board (PCB), just as we did in the course "electronics". But having gotten to an early start of the next part of the project, the software bit – I realized that time was sparse, and decided to trade the hours of producing an PCB with learning Android. Assuming 2-4 hours spent producing the board, an hourly wage of 160 NOK (simulated for the sake of the project) more than justified ordering a PCB online. I therefore designed the board and produced Gerber files, which I sent to a factory in Germany, receiving a PCB after 3 weeks for 180 NOK, ready for soldering.

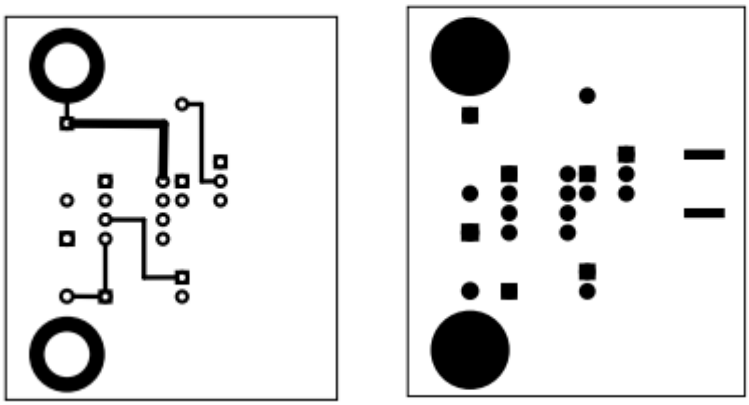


FIGURE 9 - ETCH MASKING FOR PCB PRODUCTION

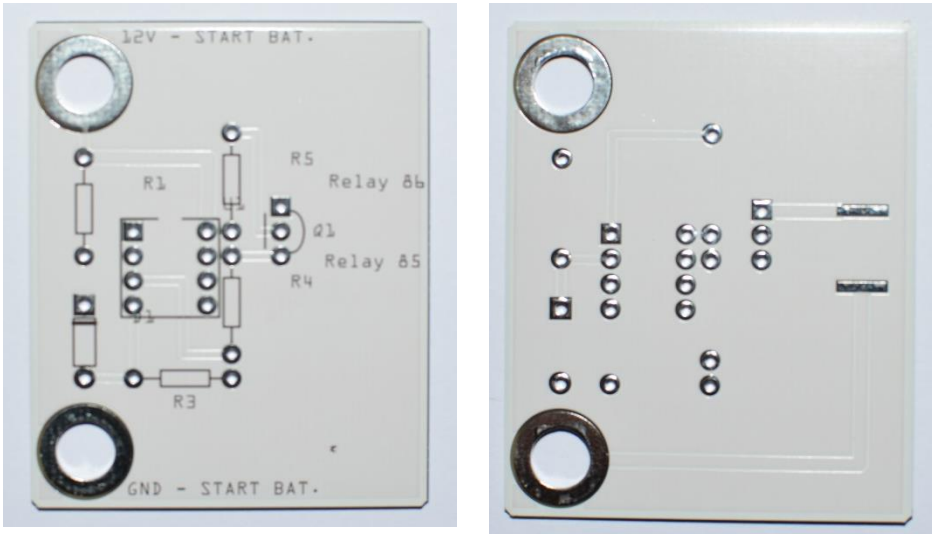


FIGURE 10 - PICTURE OF PCB PRE SOLDERING

CONNECTING SENSORS

The physical connection of the sensors to the microcontroller is straightforward. The GPS module communicates over UART – serial communication. The fuel sensor, on the other hand, sends its signal in the form of pulses of the V_{CC}. To ensure a stable readout of the sensor, I connected a pull-up resistor to the output signal, ensuring that the signal will not float between V_{CC} and ground.

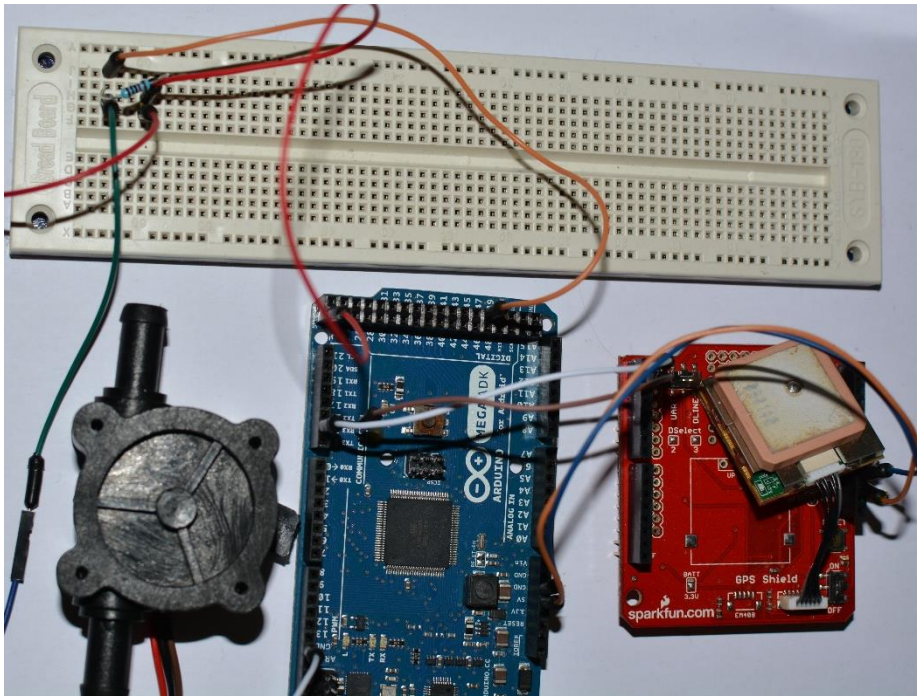


FIGURE 11 - SENSORS AND MICROCONTROLLER

PROGRAMMING THE MICROCONTROLLER

ARDUINO IDE

The ATmega2560 is programmable in the C language, and suited on the Arduino ADK it is also preloaded with an Arduino bootloader. This opens up for programming using Arduino libraries, which can come in useful for connecting to different types of accessories. For the programming, I used the Arduino IDE (Figure 12), which is a lightweight IDE supporting only basic serial communication to the connected Arduino boards, and no auto-completing options. Some syntax highlighting is available.

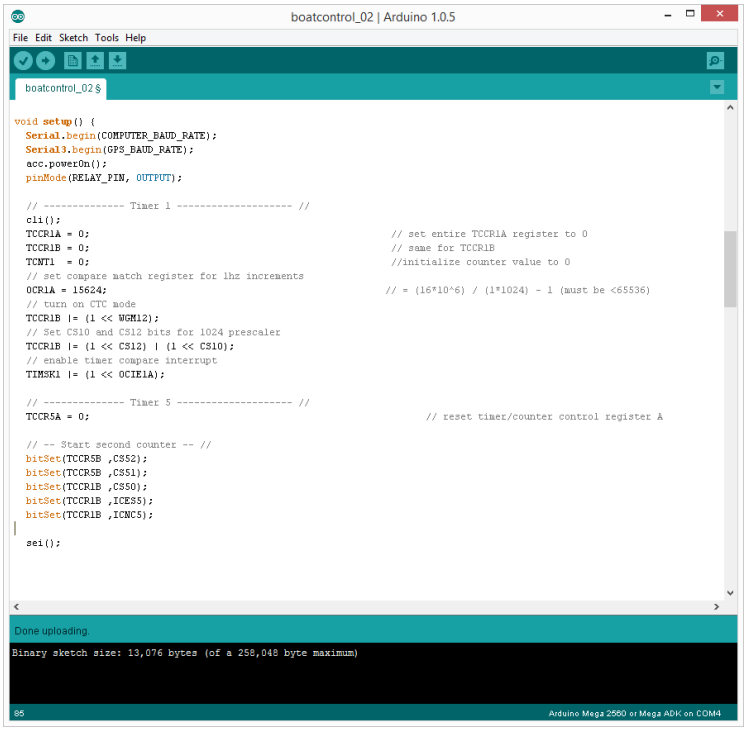


FIGURE 12 - ARDUINO IDE

ARDUINO SOURCE CODE

The Arduino source code is almost identical to the C/C++ language. A few libraries help beginners into programming, dividing the lifecycle of a program into respectively setup and loop. The setup runs once, and the loop runs indefinitely, until the power is cut or the reset button in pressed. Preceding the setup is usually definitions and global variable declarations.

During the setup, the microcontroller's registers is configured. I have used two different methods of setting the registers, both the traditional way – and a perhaps easier way, using the Arduino libraries.

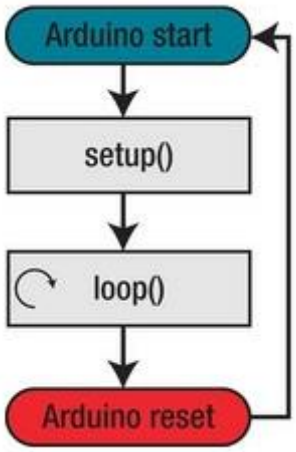


FIGURE 13 – GENERIC ARDUINO LIFECYCLE

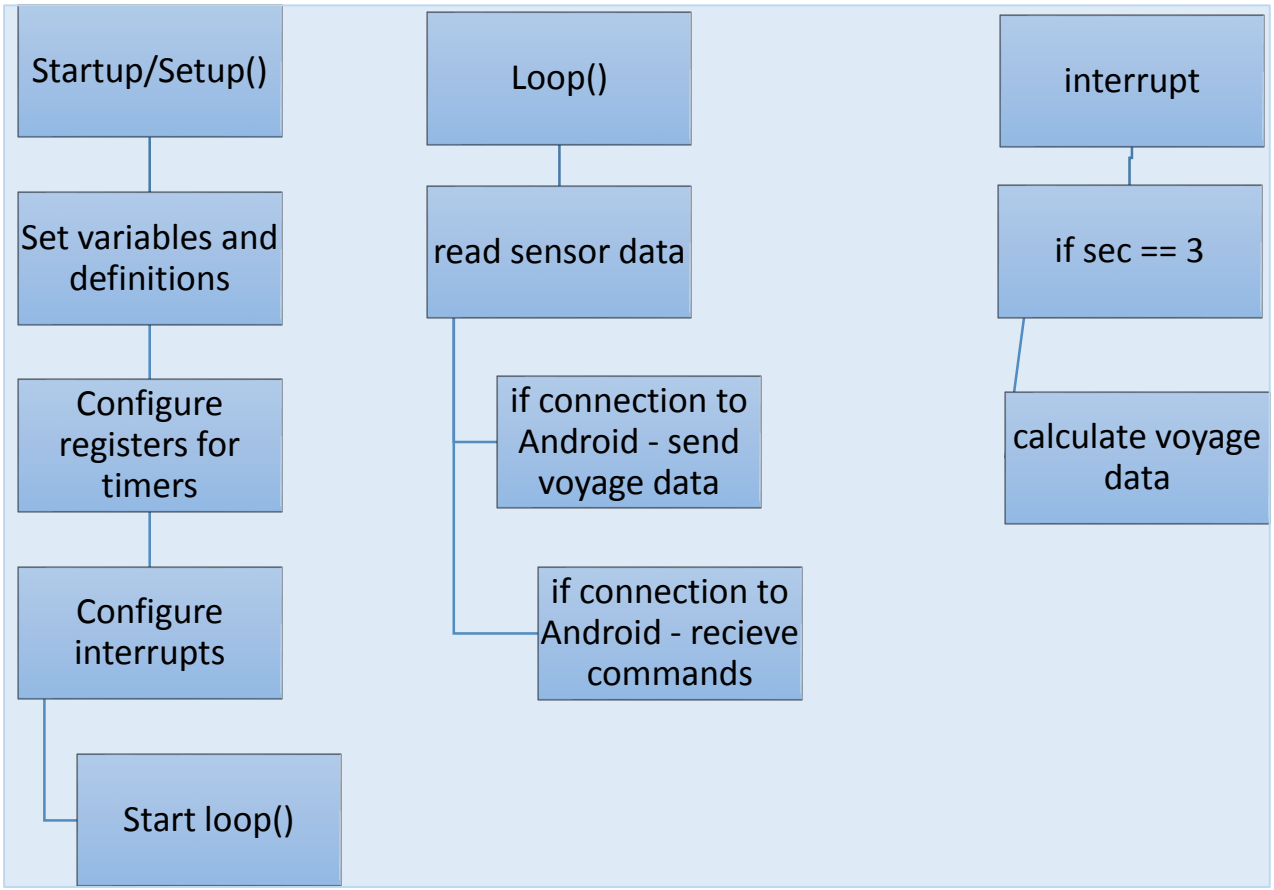


FIGURE 14 - ARDUINO CODE FLOWCHART

REGISTERS AND INTERRUPTS

To be able to count seconds I activate Timer 1 on the microcontroller. This triggers an interrupt each second according to the register configuration seen in Appendix 4: Arduino source code, under Timer 1. This timer is configured using "normal" register manipulation used in C / C++. These seconds is then again the trigger for calculating voyage data, like fuel consumption.

To measure the number of pulses received from the fuel flow sensor, using the less amount of processing power, I decided to do it all in hardware. I configured Timer 5 as a counter, but with an "external clock" – the clock being the pulses from the sensor. This gives an accurate reading, without slowing the processor down. This timer is configured using Arduino library for "bitSetting".

GPS READINGS

For reading the NMEA protocol of the UART over Serial3-connection to the GPS receiver, I use a library called TinyGPSPlus. This library has a number of methods, where I use the parsing of the NMEA protocol, extracting the speed in knots. One knot is for the record one nautical mile per hour, which again is 1852 meters an hour.

I first check if the receiver is available, and then if it has a valid signal. If the controller loses valid GPS signal for more than 5 seconds, a warning is sent.

ARDUINO – ANDROID COMMUNICATION

This has been where the most time has been spent. Getting a reliable communication protocol between the two units, and languages, C and Java/android. Luckily, there is a library helping to establish the formal connection, letting the Android framework recognize the Arduino unit as a piece of valid accessory. This library, "AndroidAccessory", requires another two libraries, "Usb" and "Max3421e" to function. The configuration is simple, you just have to define some description of the accessory, which has to be the same as in the Android application trying to connect to the accessory.

Once this formal link was up, I could start working on a simple communication protocol for transferring data both ways between the platforms. As the producers of the ADK suggested, I chose to go with a message protocol consisting of an array of bytes. Each byte in the array has a predefined set of values it can take; each has its own meaning.

Byte[0]	**Byte[1]**	**Byte[2]**	**Byte[…]**	**Byte[n]**
COMMAND / READ	TARGET	VALUE	VALUE	VALUE / CHECKSUM

FIGURE 15 - MESSAGE PROTOCOL

Byte[0] : This byte tells the receiver what sort of message that follows, if it's a command or a reading.
Byte[1] : If the message is a reading of values, this tells which value it is. If it's a command, it will tell the receiver which pin the following values are for.
Byte[2] : The following bytes are the values, either read or for the command. The final byte can be implemented as a checksum, but since this project relies on cabled communication, it seems redundant and is instead kept open for a future release.

Just sending a simple on/off message does not require more than three bytes. However, in the case of transmitting a value larger than this, the value has to be broken down before transmission, and reassembled on the receiving end. This is being done using the operation called bit shifting ">>".

I.e. The integer 276 is in its binary form: 00000000 00000000 00000001 00010100.

To split this up into four bytes we write the following code:

```
Byte[2] = (byte) (integer >> 24);
Byte[3] = (byte) (integer >> 16);
Byte[4] = (byte) (integer >> 8);
Byte[5] = (byte) (integer);
```

This results in the following:

```
Byte[2] = 0b00000000
Byte[3] = 0b00000000
Byte[4] = 0b00000001
Byte[5] = 0b00010100
```

Now the bytes are ready for transmission. On the receiving end, the unit has to add these bytes. Simply adding them will give the number 21. Therefore, we have to bit shift them back into place.

```
Integer = ((byte[2] & 0xFF) << 24 + (byte[3] & 0xFF) << 16 + (byte[4] & 0xFF) << 8 + (byte[5] & 0xFF) );
```

If a decimal is required, this can be accomplished by just multiplying by ten, or a hundred before bit shifting, and on the receiving end dividing by the same number.

The source code can be found in Appendix 4: Arduino source code.

SOFTWARE

In this project, the software term is used for the Android-application.

LEARNING

From the second year course "Informatics", I have learned basic Java programming. This is the foundation of the software division of this project, consisting of Android programming. Never the less, I spent many hours going through the basics of Android, to get the fundamentals of the architecture and how to program. For this I used online references, both free and one paid course at udemy.com; "Learn Android Programming in Java" – by John Purcell.

HISTORY

Android is a Linux based operating system introduced by Google in 2007 for mobile devices. Android promotes open standards in both software as well as hardware. Android has received critics for being fragmented, meaning it has to many versions on the market at the same time. This makes it for developers to balance its user groups to the latest functionality. On this day the latest version is Android 4.4, codenamed KitKat.

ANDROID

Android is as previously mentioned a Linux based OS. It consist of a Linux kernel, stacked with libraries and APIs written in C. On top of this, you find the applications and application framework, which is based on Java, supported by XML.

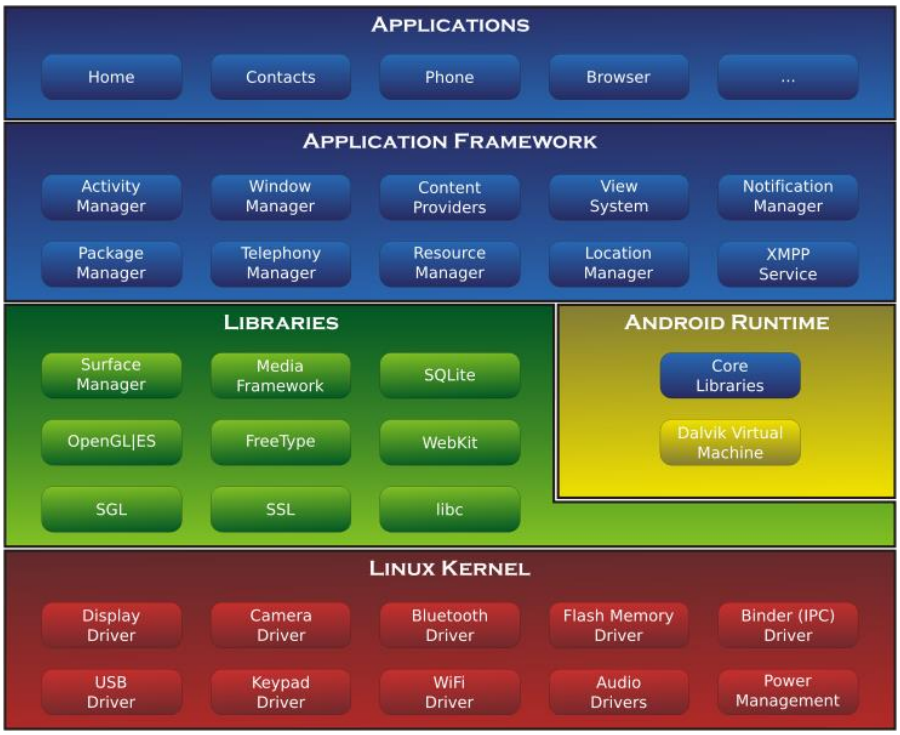


FIGURE 16- ANRDOID ARCHITECTURE(WIKIPEDIA)

This projects application is using amongst others the USB Driver for connecting to the microcontroller.

ECLIPSE – IDE

For the programming, I used the Eclipse IDE. Eclipse can be configured for a lot of different inputs, amongst others the SDK(Software development kit) for Android. This can be installed as a package, giving the user a ready to code program in the matter of minutes (or the time it takes to download).

Eclipse also serves as a debugging platform, and will receive debugging-information for either an emulated devices or a connected mobile device. This functionality has been tricky to use in this project, since the mobile device I was programming for, needed to be connected to the microcontroller, occupying the only microUSB port. This made progress slower than usual, since debugging is a quick way for making progress in a development scenario.

PRODUCTION

I will not go into a complete review of the source code here. The code is attached and heavily commented.

The programming was done systematically, function by function. As one sensor was added to the microcontroller, the receiving end was programmed to the Android application. Of course starting with the communication protocol described in the chapter “Arduino – Android communication”.

In general, an Android application has some inherited functions describing the state of the app in relation to the phone. There is a main function, “onCreate” for the first start of the app, along with “onResume”, “onPause” and “onDestroy”, which makes up the baseline for further development., see Figure 17 - Android flowchart.The most challenging was as I mentioned in the Arduino-section, to get the clean two-way communication up and running. In the Android system this included creating objects of BroadcastRecievers, UsbAccesories, FileDescriptors and in/out-putstreams. All of

which has to be managed and closed /hibernated in the right way depending on the state of the phone and/or accessory. More details is to be found in Appendix 4: Arduino source code

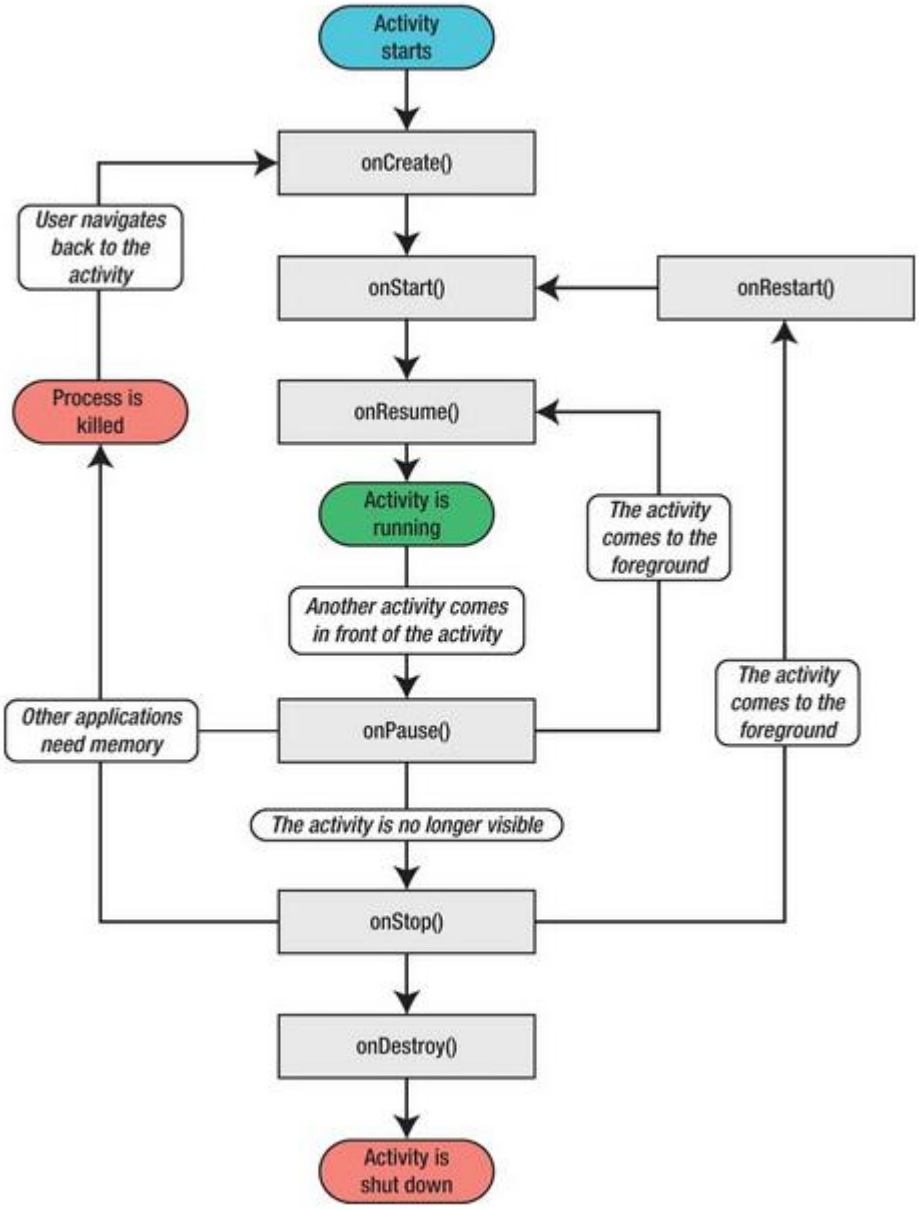


FIGURE 17 - ANDROID FLOWCHART

FACTORY ACCEPTANCE TEST - FAT

Having a valid an thorough test is paramount for every project, to keep service costs at a minimum and to keep the customers happy. In a small scale project like this, testing is done along the way, all the time, and one can say that a FAT is a bit over the top. Nevertheless I wanted to implement a simple setup for a FAT, for the sake of running this as a project in the spirit of the “Project planning”-course.

By the time I’m writing this, it is November and the boat in subject is out of the water. This is really not a problem, one can connect the engine up with a water hose and run the test as if it was alongside. The challenge, I found out, was that the marina has closed down the watering-system due to the coming witer and sub zero degrees. Therefore, I will attatch the simplesamle FAT I made for the project, unable to run it until spring arrive.

MINUTES OF MEETING

As described in the course information, minutes of meeting should be presented for each meeting. We held in total of four status meetings, from the 24th of September until the 5th of November. The attendees were the same in every meeting, Mrs. Tyssø and myself. All the meetings were held in Mrs. Tyssø's office. I regret the lack of MoM-templates, but I will reproduce the content in the following section. Due to sickness, a meeting was canceled. The meetings were short and precise.

MEETING NUMBER 1 - 24TH SEPTEMBER

This meeting was the first and only meeting where the MS Project was still functioning. I brought a status overview (Appendix 6: Status meeting nr. 1 attachment), where the current progress was depictured. This meeting also served as the first face to face meeting during the project, and was more or less a run through and presentation of the project.

MEETING NUMBER 2 – 7TH OCTOBER

Work done: Hardware finished, lacking the production of the PCB. Parallel learning of Arduino.
Until next meeting: Started the software production, started making report. Notes and pictures taken along the way.

MEETING NUMBER 3 – 4TH NOVEMBER

Work done: Software under way, had to reprioritize the making of PCB with more hours in learning Arduino.
Until finish: Finish report, if time add functionality and fixes to application.

CONCLUSION AND POTENTIAL

Every project has some what do go on its potential, as well as this. There is a few things that is apparent where time has not been sufficient, and future releases has to implement the fixes. There is also some functionalities I would like to see in future releases, which has been outside the scope of this project. The focus has been in this project to establish a framework consisting of sensors, processing power, a graphical user interface and not least the knowledge of putting this variety of technology together into a product. The communication between the elements are in place, ranging from UART, PPM to the "home made" communication protocol between the microcontroller and the GUI. The voltage sensing relay is produced and long term observation will tell if the Zener diode does the trick under different operation scenarios, or has to be replaced by another reference.

SHORTCOMMINGS

- The GUI is not pretty. It serves as showing reliable information from the microcontroller and controlling the relay. This has to be fixed in a future release.
- Mounting. Even though not specifically mentioned, casings and mounting for the equipment is essential before operation at sea.

POTENTIAL

- Further control of the 12V auxiliary system on board. I have to buy a couple of relays, replacing the switches controlling i.e. the lanterns and the radio. Using the on board Android luminescence sensor lanterns can easily be configured to turn on at dusk.
- Anchor watch. With the GPS in place, an alarm can sound if the boat is at anchor but starts to drift.
- Alarms. If low on fuel, low battery voltage, etc.
- Ships log. Using GPS voyage data, a database can contain trip information, containing the fuel consumption, average speed, etc.
- Theft alarm.
- Using the android devices GSM-module, unauthorized access to the boat can be warned over SMS, Twitter, etc.
- Replacing the USB cable with XBee. Also using a XBee mesh for integrating engine diagnostics.
- As mentioned, with the framework in place – the imagination and time is the only thing limiting further development.

CONCLUSION

In addition to the previously mentioned, I have to state that this has been a project full of learning. It has taken the essence of the previous subjects, whilst then presented inside a closed frame, now put together independently with no external limitations. Having this oversight and to plan it all is well worth the experience. The availability of consumer grade electronics today has never been better, and this project has re-motivated me for further development on a hobby basis, as well as a professional in the future.

APPENDIX 1: GANTT-CHART

Task Name	Work	Baseline
Oppstart	0 hrs	0 hrs
Planlegging	4,5 hrs	0 hrs
Hardware	**40,5 hrs**	**0 hrs**
Hardware design	9 hrs	0 hrs
Hardware produksjon prototype	9 hrs	0 hrs
Hardware prototype test	9 hrs	0 hrs
Hardware produkt produksjon	9 hrs	0 hrs
Hardware produkt test	4,5 hrs	0 hrs
Software	**94,5 hrs**	**0 hrs**
Software design	9 hrs	0 hrs
Software læring	27 hrs	0 hrs
Software produksjon	45 hrs	0 hrs
Software test	13,5 hrs	0 hrs
FAT produkt	9 hrs	0 hrs
Ferdig	0 hrs	0 hrs

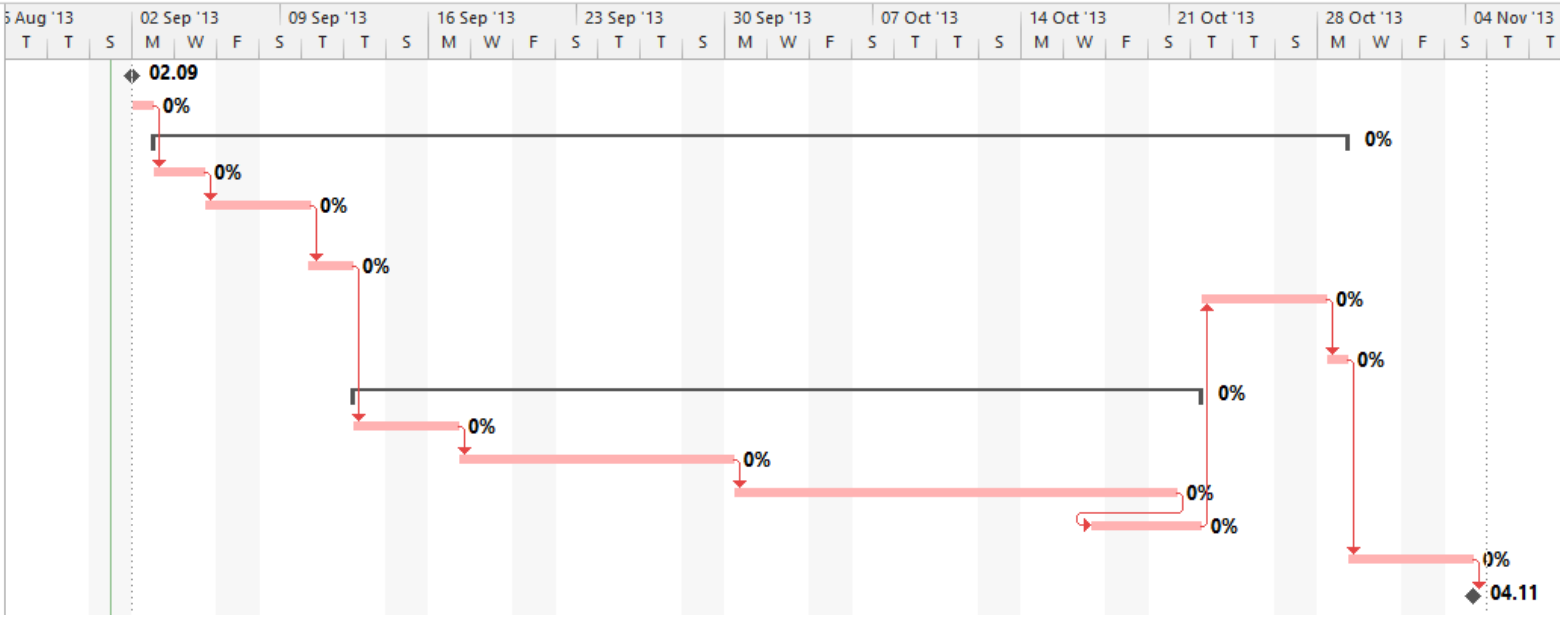


FIGURE 18 - GANTT CHART

APPENDIX 2: PROJECT PRESENTATION (IN NORWEGIAN)

«STYRINGS- OG OVERVÅKINGSSYSTEM FOR FRITIDSBÅT»

Elektroprosjekt, høsten 2013.

av
Emil Niclas Hofseth

01.09.2013

INNLEDNING OG PRESENTASJON

Valgemnet «Elektroprosjekt (LV388E)» har som mål at prosjektet skal gi studenten en fordypning innen et selvvalgt emne. Selve inkluderingen av emnet i min fremdriftsplan ble foretatt våren 2013 uten da å fastslå selve prosjektet. Sommeren har derfor blitt benyttet til å finne et passende prosjekt som vil oppfylle målet til emnet, samt å være inspirerende, lærerikt og nyttig for videre utdanning og arbeid. Da ingeniørstudiet innen elektronikk og informasjonsteknologi inneholder både hardware- og software-utdanning, ønsker jeg å bygge på begge disse fagfeltene og ha et prosjekt som knytter sammen det vi har lært til nå, fra elektroniske kretser, til lav- og høy-nivå programmering.

Undertegnende eier selv en båt, drevet av en utenbords-motor med tilknyttet batteri og en enkel elektrisk krets for tilkobling av forbruks-artikler. I egen og andres hverdag er det et økende antall enheter vi er mer eller mindre avhengige av som drives av batterier; eksempelvis mobiltelfoner, GPS og andre bærbare datamaskiner. Med dagens batteriteknologier krever disse også en forholdsvis hyppig ladefrekvens. 12V-anlegget ombord i båten kan fungere som en kilde til lading av disse enhetene, som en tilleggsfunksjon til det primære, som er å starte utenbords-motoren.
Ulempen med 12V-anlegget er at det ikke har noen indikator på batteriets status, og en risikerer derfor at det ikke er nok spenning til å starte motoren på grunn av overdrevent forbruk. En løsning er å tilknytte et ytterligere batteri, som kan betjene forbruket. Dette batteriet må også tilkobles motorens generator, slik at også det blir ladet opp under gange, slik som startbatteriet. I tillegg må startbatteriet frakobles når generatoren ikke leverer ladning, slik at det ikke tappes og er klar til bruk når motoren igjen skal startes.
Denne funksjonaliteten skal drives autonomt, være drift- og brannsikker, samt virke under alle relevante temperaturer. Denne delen av prosjektet ønskes gjennomført av hardware-komponenter uten aktiv prosessering i form av mikroprosessorer eller lignende. Om det er gjennomførbart, ønsker jeg å se på mulighetene for å ta ut diagnostikk-data fra motoren direkte, og overføre trådløst via Zigbee til mikrokontrolleren, men det kommer an på hvor avanserte protokoller som benyttes til denne diagnostikken.

Videre ønskes det en måte for føreren å lese av forskjellige statusverdier tilhørende båten. Eksempler på dette er batterispenning, drivstoff-forbruk og hastighet. Det er også ønskelig å kunne overstyre automatikken i sammenkoblingsmekanismen til batteriene, nevnt over. Dette ønsker jeg å gjennomføre ved å bygge videre på emnet «Digitale systemer 2» som fokuserer på C-programmering. Jeg tilegnet meg der kunnskaper om mikrokontrollere og ser for meg en mikrokontroller tilknyttet forskjellige sensorer for å å lese av og lagre verdier. Her kan også kombinasjoner av sensorer som f.eks. drivstoff-forbruk og hastighet kombineres for å gi

en måling på økonomiske hastigheter, potensiell rekkevidde og annen informasjon som er svært nyttig når man ferdes på havet.

For å presentere dette på en enkel måte, ønsker jeg her å bygge videre på emnet «Informatikk», hvor vi lærte grunnleggende Java-programmering. Selv er jeg bruker av Android-økosystemet på mobile enheter, som baserer seg på C og Java, og jeg ser for meg å programmere en applikasjon som presenterer informasjonen mikrokontrolleren henter inn - samt mulighet for å grafisk styre f.eks. Batterisammenkoblingen, 12V-systemet og herunder f.eks. lanternebruk, via mikrokontrolleren. Jeg har aldri tidligere programmert en applikasjon til Android før, så dette krever en god del opplæring. Som ressurs vil jeg benytte meg av internett-kurs og litteratur jeg selv har anskaffet.

For en skisse over forestilt system se vedlegg nr. 2.

FREMDRIFT OG PLANLEGGING

Som planleggingsverktøy vil jeg benytte meg av Microsoft Project. Både for initiell planlegging, men også for gjennomføring, rapportering og ressursadministrasjon. Det planleges med 150-160 arbeidstimer, fordelt på de forskjellige oppgavene. Prosjektet med rapport forventes ferdigstilt til mandag 4. november. Se vedlegg 1 for detaljer. Rapportering av fremdrift/statusmøte gjøres ihht. emneplan og nærmere avtale senest hver fjortende dag med veileder.

MATERIALER OG KOSTNADER

Da planleggingen av prosjektet har foregått tidvis gjennom sommeren, har også materialer blitt anskaffet på forhånd, da dette sparer prosjektet for mye tid. Siden produktet ønskes å beholdes vil også alle utgifter dekkes av undertegnende. Kostnader for materialene samt arbeidstimer vil også medregnes i prosjektkostnadene og inkluderes i rapportene

APPENDIX 3: ANDROID SOURCE CODE

Java-code only.

```
1.  package no.priv.enh.android.boatcontrol;
2.
3.  import java.io.FileDescriptor;
4.  import java.io.FileInputStream;
5.  import java.io.FileOutputStream;
6.  import java.io.IOException;
7.
8.  import android.app.Activity;
9.  import android.app.PendingIntent;
10. import android.content.BroadcastReceiver;
11. import android.content.Context;
12. import android.content.Intent;
13. import android.content.IntentFilter;
14. import android.os.AsyncTask;
15. import android.os.Bundle;
16. import android.os.ParcelFileDescriptor;
17. import android.util.Log;
18. import android.widget.CompoundButton.OnCheckedChangeListener;
19. import android.widget.CompoundButton;
20. import android.widget.ProgressBar;
21. import android.widget.TextView;
22. import android.widget.ToggleButton;
23.
24. import com.android.future.usb.UsbAccessory;
25. import com.android.future.usb.UsbManager;
26.
27. public class BoatControlActivity extends Activity {
28.
29.     private static final String TAG = BoatControlActivity.class.getSimpleName();
    // Tag for simple debugging
30.
31.     private PendingIntent mPermissionIntent;
32.     private static final String ACTION_USB_PERMISSION = "com.android.example.USB_PERMIS
    SION";
33.     private boolean mPermissionRequestPending;
34.
35.     private UsbManager mUsbManager;
36.     private UsbAccessory mAccessory;
37.     private ParcelFileDescriptor mFileDescriptor;
38.     private FileInputStream mInputStream;
39.     private FileOutputStream mOutputStream;
40.
41.     // ----------  Global variables / commands -------------- //
42.     private static final byte COMMAND_READING = 0x1;
43.     private static final byte COMMAND = 0x3;
44.     private static final byte TARGET_SPEED = 0x5;
45.     private static final byte TARGET_NM_PER_LITER = 0x6;
46.     private static final byte TARGET_FUEL_CONSUMPTION = 0x7;
47.     private static final byte TARGET_RELAY_PIN = 0x2;
48.     private static final byte TARGET_BATTERY_ONE_PIN = 0x8;
49.     private static final byte TARGET_BATTERY_TWO_PIN = 0x9;
50.
51.     // ------------  Graphical elements ---------------------- //
52.     private TextView speedTextView;
53.     private ProgressBar speedProgressBar;
54.     private TextView nmPerLiterTextView;
55.     private ProgressBar nmPerLiterProgressBar;
56.     private TextView fuelConsumptionTextView;
57.     private ProgressBar fuelConsumptionProgressBar;
```

```
58.     private TextView batteryOneTextView;
59.     private ProgressBar batteryOneProgressBar;
60.     private TextView batteryTwoTextView;
61.     private ProgressBar batteryTwoProgressBar;
62.
63.     private ToggleButton relayToggleButton;
64.     private ToggleButton auxToggleButton;
65.     private ToggleButton radioToggleButton;
66.
67.     @Override
68.     public void onCreate(Bundle savedInstanceState) {                           // onCr
    eate - when app first starts, remembers last state
69.         super.onCreate(savedInstanceState);
70.
71.         mUsbManager = UsbManager.getInstance(this);                             // UsbM
    anager manages all interaction with the USB port, enumerates
72.         mPermissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(     // devi
    ces and requests and checks permissions
73.                 ACTION_USB_PERMISSION), 0);
74.         IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
75.         filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED);
76.         registerReceiver(mUsbReceiver, filter);
77.
78.         setContentView(R.layout.main);
    // Enables layout and initializes graphical elements
79.         speedTextView = (TextView) findViewById(R.id.speed_text_view);
80.         speedProgressBar = (ProgressBar) findViewById(R.id.speed_bar);
81.         nmPerLiterTextView = (TextView) findViewById(R.id.nm_per_liter_text_view);
82.         nmPerLiterProgressBar = (ProgressBar) findViewById(R.id.nm_per_liter_bar);
83.         fuelConsumptionTextView = (TextView) findViewById(R.id.fuel_consumption_text_vi
    ew);
84.         fuelConsumptionProgressBar = (ProgressBar) findViewById(R.id.fuel_consumption_b
    ar);
85.         batteryOneTextView = (TextView) findViewById(R.id.battery_one_text_view);
86.         batteryOneProgressBar = (ProgressBar) findViewById(R.id.battery_one_bar);
87.         batteryTwoTextView = (TextView) findViewById(R.id.battery_two_text_view);
88.         batteryTwoProgressBar = (ProgressBar) findViewById(R.id.battery_two_bar);
89.
90.         relayToggleButton = (ToggleButton) findViewById(R.id.relay_toggle_button);
91.         relayToggleButton
92.                 .setOnCheckedChangeListener(toggleButtonCheckedListener);
93.         auxToggleButton = (ToggleButton) findViewById(R.id.aux_toggle_button);
94.         auxToggleButton.setOnCheckedChangeListener(toggleButtonCheckedListener);
95.         radioToggleButton = (ToggleButton) findViewById(R.id.radio_toggle_button);
96.         radioToggleButton
97.                 .setOnCheckedChangeListener(toggleButtonCheckedListener);
98.
99.     }
100.
101.        @Override
102.        public void onResume() {
103.            super.onResume();
104.
105.            if (mInputStream != null && mOutputStream != null) {
106.                return;
107.            }
108.
    // Checks if in/output stream is still active, checks permission
109.            UsbAccessory[] accessories = mUsbManager.getAccessoryList();
    // and reestablishes connection to accessory
110.            UsbAccessory accessory = (accessories == null ? null : accessories[0]);
111.            if (accessory != null) {
112.                if (mUsbManager.hasPermission(accessory)) {
113.                    openAccessory(accessory);
114.                } else {
115.                    synchronized (mUsbReceiver) {
```

```
116.                         if (!mPermissionRequestPending) {
117.                             mUsbManager.requestPermission(accessory,
118.                                     mPermissionIntent);
119.                             mPermissionRequestPending = true;
120.                         }
121.                     }
122.                 }
123.             } else {
124.                 Log.d(TAG, "mAccessory is null");
125.             }
126.         }
127.
128.         @Override
129.         public void onPause() {
130.             super.onPause();
131.             closeAccessory();
132.         }
133.
134.         @Override
135.         public void onDestroy() {                                    // removes acces
    sory from register
136.             super.onDestroy();
137.             unregisterReceiver(mUsbReceiver);
138.         }
139.
140.         OnCheckedChangeListener toggleButtonCheckedListener = new OnCheckedChangeListene
    r() {           // A listener for the relay togglebutton
141.
142.             @Override
143.             public void onCheckedChanged(CompoundButton buttonView,
144.                     boolean isChecked) {
145.                 if (buttonView.getId() == R.id.relay_toggle_button) {
146.
147.                     new AsyncTask<Boolean, Void, Void>() {
148.
149.                         @Override
150.                         protected Void doInBackground(Boolean... params) {
151.                             sendRelaySwitchCommand(TARGET_RELAY_PIN, params[0]);
152.                             return null;
153.                         }
154.                     }.execute(isChecked);
155.                 }
156.
157.             }
158.         };
159.
160.         private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver() {
      // Anonymous inner class for the BroadCastReceiver. Overwrites onrecieve
161.             @Override
      // Broadcastreceiver calls the open/close accessory method.
162.             public void onReceive(Context context, Intent intent) {
163.                 String action = intent.getAction();
164.                 if (ACTION_USB_PERMISSION.equals(action)) {
165.                     synchronized (this) {
166.                         UsbAccessory accessory = UsbManager.getAccessory(intent);
167.                         if (intent.getBooleanExtra(
168.                                 UsbManager.EXTRA_PERMISSION_GRANTED, false)) {
169.                             openAccessory(accessory);
170.                         } else {
171.                             Log.d(TAG, "permission denied for accessory "
172.                                     + accessory);
173.                         }
174.                         mPermissionRequestPending = false;
175.                     }
176.                 } else if (UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action)) {
177.                     UsbAccessory accessory = UsbManager.getAccessory(intent);
```

```
178.                     if (accessory != null && accessory.equals(mAccessory)) {
179.                         closeAccessory();
180.                     }
181.                 }
182.             }
183.         };
184.
185.         private void openAccessory(UsbAccessory accessory) {
        // Gets a filedescriptor for the accessory, manages in/outputstream.
186.             mFileDescriptor = mUsbManager.openAccessory(accessory);
        // If successfull, starts new thread
187.             if (mFileDescriptor != null) {
188.                 mAccessory = accessory;
189.                 FileDescriptor fd = mFileDescriptor.getFileDescriptor();
190.                 mInputStream = new FileInputStream(fd);
191.                 mOutputStream = new FileOutputStream(fd);
192.                 Thread thread = new Thread(null, commRunnable, TAG);

193.                 thread.start();
194.                 Log.d(TAG, "accessory opened");
195.             } else {
196.                 Log.d(TAG, "accessory open fail");
197.             }
198.         }
199.
200.         private void closeAccessory() {
201.             try {
202.                 if (mFileDescriptor != null) {
203.                     mFileDescriptor.close();
204.                 }
205.             } catch (IOException e) {
206.             } finally {
207.                 mFileDescriptor = null;
208.                 mAccessory = null;
209.             }
210.         }
211.
212.         public void sendRelaySwitchCommand(byte target, boolean isSwitchedOn) {
            // Method for preparing and sending commands to Arduino
213.             byte[] buffer = new byte[3];
214.             buffer[0] = COMMAND;
215.             buffer[1] = target;
216.             if (isSwitchedOn) {
217.                 buffer[2] = 0x1;
218.
219.             } else {
220.                 buffer[2] = 0x0;
221.             }
222.             if (mOutputStream != null) {
223.                 try {
224.                     mOutputStream.write(buffer);
225.                 } catch (IOException e) {
226.                     Log.e(TAG, "write failed", e);
227.                 }
228.             }
229.         }
230.
231.         Runnable commRunnable = new Runnable() {                                    // T
    hread for handling the in/output communication , both receiving and
232.                                                                                     // b
    uilding/sending byte arrays. Elseif determines type of command/value
233.             @Override
234.             public void run() {
235.                 int ret = 0;
236.                 byte[] buffer = new byte[6];
237.
```

```
238.                while (ret >= 0) {
239.                    try {
240.                        ret = mInputStream.read(buffer);
241.                    } catch (IOException e) {
242.                        Log.e(TAG, "IOException", e);
243.                        break;
244.                    }
245.
246.                    switch (buffer[0]) {
247.                    case COMMAND_READING:
248.
249.                        if (buffer[1] == TARGET_SPEED) {
250.                            final int speed = ((buffer[2] & 0xFF) << 24)
251.                                    + ((buffer[3] & 0xFF) << 16)
252.                                    + ((buffer[4] & 0xFF) << 8)
253.                                    + (buffer[5] & 0xFF);
254.                            runOnUiThread(new Runnable() {
255.
256.                                @Override
257.                                public void run() {
258.                                    speedProgressBar.setProgress(speed);
259.                                    speedTextView.setText(getString(
260.                                            R.string.speed_text, speed));
261.                                }
262.                            });
263.                        } else if (buffer[1] == TARGET_NM_PER_LITER) {
264.                            final int nmPerLiter = ((buffer[2] & 0xFF) << 24)
265.                                    + ((buffer[3] & 0xFF) << 16)
266.                                    + ((buffer[4] & 0xFF) << 8)
267.                                    + (buffer[5] & 0xFF);
268.                            runOnUiThread(new Runnable() {
269.
270.                                @Override
271.                                public void run() {
272.                                    nmPerLiterProgressBar.setProgress(nmPerLiter);
273.                                    nmPerLiterTextView
274.                                            .setText(getString(
275.                                                    R.string.nm_per_liter_text,
276.                                                    nmPerLiter));
277.                                }
278.                            });
279.                        } else if (buffer[1] == TARGET_FUEL_CONSUMPTION) {
280.                            final int fuelConsumption = ((buffer[2] & 0xFF) << 24)
281.                                    + ((buffer[3] & 0xFF) << 16)
282.                                    + ((buffer[4] & 0xFF) << 8)
283.                                    + (buffer[5] & 0xFF);
284.                            runOnUiThread(new Runnable() {
285.
286.                                @Override
287.                                public void run() {
288.                                    fuelConsumptionProgressBar
289.                                            .setProgress(fuelConsumption);
290.                                    fuelConsumptionTextView.setText(getString(
291.                                            R.string.fuel_consumption_text,
292.                                            fuelConsumption));
293.                                }
294.                            });
295.                        } else if (buffer[1] == TARGET_BATTERY_ONE_PIN) {
296.                            final float batteryOneVoltageFloat = ((buffer[2] & 0xFF) <<
    24)
297.                                    + ((buffer[3] & 0xFF) << 16)
298.                                    + ((buffer[4] & 0xFF) << 8)
299.                                    + (buffer[5] & 0xFF) / 10;
300.                            final int batteryOneVoltage = (int) batteryOneVoltageFloat;

301.                            runOnUiThread(new Runnable() {
```

```
302.
303.                                 @Override
304.                                 public void run() {
305.                                     batteryOneProgressBar
306.                                             .setProgress(batteryOneVoltage);
307.                                     batteryOneTextView.setText(getString(
308.                                             R.string.battery_one_text,
309.                                             batteryOneVoltageFloat));
310.                                 }
311.                             });
312.                         } else if (buffer[1] == TARGET_BATTERY_TWO_PIN) {
313.                             final float batteryTwoVoltageFloat = ((buffer[2] & 0xFF) <<
    24)
314.                                     + ((buffer[3] & 0xFF) << 16)
315.                                     + ((buffer[4] & 0xFF) << 8)
316.                                     + (buffer[5] & 0xFF) / 10;
317.                             final int batteryTwoVoltage = (int) batteryTwoVoltageFloat;

318.                             runOnUiThread(new Runnable() {
319.
320.                                 @Override
321.                                 public void run() {
322.                                     batteryTwoProgressBar
323.                                             .setProgress(batteryTwoVoltage);
324.                                     batteryTwoTextView.setText(getString(
325.                                             R.string.battery_two_text,
326.                                             batteryTwoVoltageFloat));
327.                                 }
328.                             });
329.                         }
330.                         break;
331.
332.                     default:
333.                         Log.d(TAG, "unknown msg: " + buffer[0]);
334.                         break;
335.                     }
336.                 }
337.             }
338.         };
339.     }
```

APPENDIX 4: ARDUINO SOURCE CODE

```
// ------------------------------ Includes for ADK/USB ------------------------------ //
#include <Max3421e.h>
#include <Usb.h>
#include <AndroidAccessory.h>

// ------------------------------ Includes for GPS ------------------------------ //
#include <TinyGPS++.h>

// ------------------------------ Defining constants ------------------------------ //
#define FUEL_SENSOR_PIN 47                      // External clock pin for timer 5
#define BATTERY_ONE_PIN A0
#define BATTERY_TWO_PIN A1
#define RELAY_PIN 2
#define COMPUTER_BAUD_RATE 115200               // Serial baudrate
#define FUEL_TANK_SIZE 25
#define GPS_BAUD_RATE 4800
#define COMMAND_READING 0x1
#define COMMAND 0x3
#define TARGET_SPEED 0x5
#define TARGET_NM_PER_LITER 0x6
#define TARGET_FUEL_CONSUMPTION 0x7
#define TARGET_RELAY_PIN 0x2
#define TARGET_BATTERY_ONE_PIN 0x8
#define TARGET_BATTERY_TWO_PIN 0x9

// Identifying the accessory for the Android unit
AndroidAccessory acc("Manufacturer",
"Project04",
"Description",
"Version",
"URI",
"Serial");

// ------------------ Variables ------------------------//
volatile float pulses;                  // Counts pulses from sensor
float litersHour;                       // Estimated liters of fuel in an hour
int count;
int sec;
float liters_per_pulse = 0.0005 ;       // Liters per pulse from datasheet of sensor

int currentADCValue;

byte sntmsg[6];
byte rcvmsg[3];
int speedInKnots = 7;
float nmPerLiterFloat;
float  fuelConsumptionFloat;
float fuelRemainingFloat;
int distanceRemaining;

TinyGPSPlus gps;                        // Creates GPS

void setup() {
  Serial.begin(COMPUTER_BAUD_RATE);
  Serial3.begin(GPS_BAUD_RATE);
  acc.powerOn();
  pinMode(RELAY_PIN, OUTPUT);

  // -------------- Timer 1 -------------------- //
```

```
60.   cli();
61.   TCCR1A = 0;                                                    // set entire TCCR1A
    register to 0
62.   TCCR1B = 0;                                                    // same for TCCR1B
63.   TCNT1  = 0;                                                    // initialize counter
    value to 0
64.   // set compare match register for 1hz increments
65.   OCR1A = 15624;
66.   TCCR1B |= (1 << WGM12);                                        // turn on CTC mode
67.   TCCR1B |= (1 << CS12) | (1 << CS10);                           // Set CS10 and CS12
    bits for 1024 prescaler
68.   TIMSK1 |= (1 << OCIE1A);                                      // enable timer compar
    e interrupt
69.   // -------------- Timer 5 -------------------
    - //               // Used to measure fuel flow
70.   TCCR5A = 0;                                                    // reset timer/counte
    r control register A
71.   bitSet(TCCR5B ,CS52);                                          // 52, 51 and 50 enab
    ling external clock source on rising edge
72.   bitSet(TCCR5B ,CS51);
73.   bitSet(TCCR5B ,CS50);
74.   bitSet(TCCR5B ,ICES5);                                         // Input capture edge
    select
75.   bitSet(TCCR5B ,ICNC5);                                         // Input capture noic
    e canceler
76.   sei();
77.
78. }
79.
80. // -------------- Timer 1 interrupt -------------------- //
81. ISR(TIMER1_COMPA_vect){
82.   sec++;
83. }
84.
85. void loop() {
86.
87.   // Read from GPS-module and get speed in knots.
88.   if(Serial3.available() > 0){
89.   }
90.   if(gps.location.isValid()){
91.     speedInKnots = (int)gps.speed.knots();
92.   }
93.
94.   if (millis() > 5000 && gps.charsProcessed() < 10){
95.     Serial.println("No GPS data received: check wiring");
96.   }
97.
98.   if(sec == 3){                                                  // Pulses the last 3 se
    conds, converts to an average
99.     sec = 0;
100.       pulses = TCNT5;
101.       TCNT5 = 0;
102.       litersHour = ((pulses / 3) * liters_per_pulse) * 3600;
103.       nmPerLiterFloat = speedInKnots/litersHour;
104.       fuelConsumptionFloat += (pulses * liters_per_pulse);
105.       fuelRemainingFloat = FUEL_TANK_SIZE - fuelConsumptionFloat;
106.       distanceRemaining = (int)fuelRemainingFloat * nmPerLiterFloat;
107.     }
108.
109.     currentADCValue = analogRead(BATTERY_ONE_PIN);
110.     sendSpeed();
111.     sendNmPerLiter();
112.     sendFuelConsumption();
113.     sendVoltageBatOne();
114.     readCommand();
115.
```

```
116.    } // LOOP
117.
118.    // ------------------------- Methodes --------------------- //
119.    void sendSpeed(){
120.      if (acc.isConnected()) {                              // Checks to find Android, the
    n sets up byte array
121.        sntmsg[0] = COMMAND_READING;                        // based on bitshifting
122.        sntmsg[1] = TARGET_SPEED;
123.        sntmsg[2] = (byte) (speedInKnots >> 24);
124.        sntmsg[3] = (byte) (speedInKnots >> 16);
125.        sntmsg[4] = (byte) (speedInKnots >> 8);
126.        sntmsg[5] = (byte) speedInKnots;
127.        acc.write(sntmsg, 6);
128.        delay(100);
129.      }
130.    }
131.
132.    void sendNmPerLiter(){
133.      if (acc.isConnected()) {
134.        int nmPerLiter = (int)nmPerLiterFloat;
135.        sntmsg[0] = COMMAND_READING;
136.        sntmsg[1] = TARGET_NM_PER_LITER;
137.        sntmsg[2] = (byte) (nmPerLiter >> 24);
138.        sntmsg[3] = (byte) (nmPerLiter >> 16);
139.        sntmsg[4] = (byte) (nmPerLiter >> 8);
140.        sntmsg[5] = (byte) nmPerLiter;
141.        acc.write(sntmsg, 6);
142.        delay(100);
143.      }
144.    }
145.    void sendFuelConsumption(){
146.      if (acc.isConnected()) {
147.        int fuelRemaining = (int)fuelRemainingFloat;
148.        sntmsg[0] = COMMAND_READING;
149.        sntmsg[1] = TARGET_FUEL_CONSUMPTION;
150.        sntmsg[2] = (byte) (fuelRemaining >> 24);
151.        sntmsg[3] = (byte) (fuelRemaining >> 16);
152.        sntmsg[4] = (byte) (fuelRemaining >> 8);
153.        sntmsg[5] = (byte) fuelRemaining;
154.        acc.write(sntmsg, 6);
155.        delay(100);
156.      }
157.    }
158.    void sendVoltageBatOne(){
159.      if (acc.isConnected()) {
160.        int voltageMeasured = (int)getCurrentVoltage(currentADCValue)*10; // Multiple by
    10 to get decimal after bitshift
161.        sntmsg[0] = COMMAND_READING;
162.        sntmsg[1] = TARGET_BATTERY_ONE_PIN;
163.        sntmsg[2] = (byte) (voltageMeasured >> 24);
164.        sntmsg[3] = (byte) (voltageMeasured >> 16);
165.        sntmsg[4] = (byte) (voltageMeasured >> 8);
166.        sntmsg[5] = (byte) voltageMeasured;
167.        acc.write(sntmsg, 6);
168.        delay(100);
169.      }
170.    }
171.
172.    // Produces a command for the relay, based on a byte array from Android
173.    void readCommand(){
174.      if (acc.isConnected()) {
175.        int rcvlen = acc.read(rcvmsg, sizeof(rcvmsg), 1);
176.        if (rcvlen > 0){
177.          if (rcvmsg[0] == COMMAND){
178.            if ( rcvmsg[1] == TARGET_RELAY_PIN){
179.              byte value = rcvmsg[2];
```

```
180.              if (value == 0x1){
181.                digitalWrite(RELAY_PIN, HIGH);
182.              }
183.              else if(value == 0x0){
184.                digitalWrite(RELAY_PIN, LOW);
185.              }
186.            }
187.          }
188.        }
189.        delay(100);
190.      }
191.    }
192.
193.    // Calculates the voltage from the analog reading
194.    float getCurrentVoltage(int currentADCValue){
195.      return 5 * currentADCValue / 1024;
196.    }
```

APPENDIX 5: FACTORY ACCEPTANCE TEST FOR AUX. CONTROL SYSTEM

- This is a sample FAT for the product of the auxiliary control system produced in the subject of “Electronics project”. The system consists of five separate units; microcontroller, GPS receiver, Fuel flow sensor, Voltage sensing relay and mobile device having installed BoatControl.apk

Test	**Action**	**Expected Result**	**Observed result**	**Comments**	**Approved**
1.1	Connect Arduino ADK to Vcc, 5-12V. Connect computer via USB, and open a serial connection	Diodes lighting up, serial connection showing “No GPS signal, check wiring”			
1.2	Connect GPS reciever to Arduino ADK, make sure that the reciever has a clear view of the sky	Diode on GPS reciever turns solid red, serial communication stop showing “No GPS signal…”			
1.3	Connect wires from the fuel flow sensor attatched to the fuel hose, to the microcontroller. Attatch the microcontroller to the Arduino device using a USB cable.	Observe BoatControl starting up, if asked to set as default program – choose yes. Values should be set as numbers, not N/A.			
1.4	Connect the relay attatched between the batteries to the microcontroller. Press the “relay override” in the app	Listen for the relay connecting the two batteries. If nothing is heard, use a multimeter for measuring connection.			
1.5	Disable “relay override” and start the engine.	Observe valid readings in the app from the sensors, and verify that the relay is open,			

		charge running only to the start-battery.			
1.6	Let the engine run as you keep monitoring the start-battery voltage, as it surpasses 13.5V	Make sure the relay closes and both batteries receive charge from the generator.			
1.7	Set a steady forward speed on the boat	Observe valid GPS readings, in combination with the fuel flow. Observe the tanklevel dropping.			
1.8	Stop the engine	Observe the relay opening, disconnecting the batteries.			

APPENDIX 6: STATUS MEETING NR. 1 ATTACHMENT

PROJECT OVERVIEW

MON 02.09.13- MON 04.11.13

% COMPLETE

17%

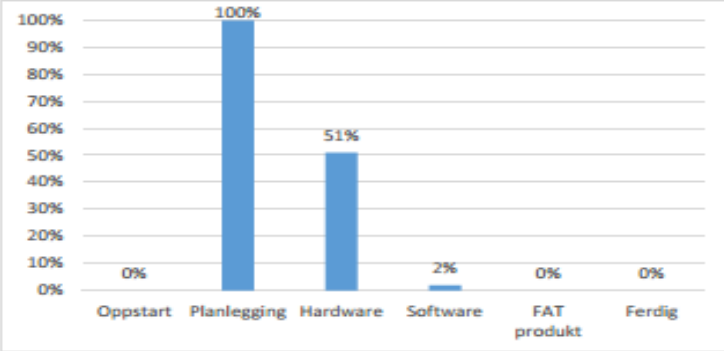
MILESTONES DUE

Milestones that are coming soon.

Name	Finish
Oppstart	Mon 02.09.13
Ferdig	Mon 04.11.13

% COMPLETE

Status for all top-level tasks. To see the status for subtasks, click on the chart and update the outline level in the Field List.



LATE TASKS

Tasks that are past due.

Name	Start	Finish	Duration	% Complete	Resource Names
Oppstart	Mon 02.09.13	Mon 02.09.13	0 days	0%	
Hardware produksjon prototype	Thu 05.09.13	Tue 10.09.13	6,5 hrs?	69%	Emil
Hardware prototype test	Tue 10.09.13	Thu 12.09.13	6 hrs?	67%	Emil
Software design	Thu 12.09.13	Tue 17.09.13	9 hrs?	11%	Emil
Software læring	Tue 17.09.13	Mon 30.09.13	27 hrs	4%	Emil