Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Odd Inge Halsos

Embedded System for Underwater Data Acquisition

Master's thesis in Electronic Systems Design
Supervisor: Bjørn B. Larsen, Jo Arve Alfredsen
July 2020

NTNU
Norwegian University of
Science and Technology

Odd Inge Halsos

Embedded System for Underwater Data Acquisition

Master’s thesis in Electronic Systems Design
Supervisor: Bjørn B. Larsen, Jo Arve Alfredsen
July 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Abstract

A small-scale buoyancy vehicle, designed to maintain a target depth between 0 and 50 meters, has potential to become a useful tool with applications related to underwater current tracking and data acquisition. A list of specifications are obtained through a use case analysis that form the design criteria for a new embedded sensor- and control system. The goal of this project is to design and implement a system that satisfy the use case requirements.

The implemented system include a PID control algorithm and interfacing to motor controller; SD card and data logging; wireless Bluetooth interfacing, to transfer log data and set vehicle configurations; and various sensors for vehicle control and data acquisition. The system is implemented in the form of a custom designed PCB, sensor modules, firmware, and various mechanical modifications. Sensors not critical for the vehicle's core functionality are not configured, instead they are installed and ready to be configured at a later stage. A terminal application is developed to interface the vehicle with a Bluetooth Low Energy enabled computer.

The vehicle is tested on land, in fresh water, and briefly in salt water. The core functionality is tested and verified; first in dry conditions, then in a freshwater tank. A freshwater lake is found suitable for PID tuning, where a set of PID coefficients are found with a system response that enable the vehicle to maintain target depth. The vehicle is tested in salt water with the same coefficient values found in the freshwater lake, where system response is different and not suitable for operation. Further testing is needed to obtain suitable system response in salt water.

Wireless air-water interface properties are investigated and tested with two antenna configurations in fresh- and salt water. A FlexPIFA is found superior to a ceramic chip monopole antenna, where reliable connection is maintained for depths down to 10 cm in fresh water and 5 cm in salt water.

Sammendrag

En små-skala oppdrftsaktuator, designet for a holde en gitt dybde mellom 0 og 50 meter, har potensial til å bli et nyttig verktøy med bruksområde relatert til kartlegging av undervannsstrømmer og innsamling av data. Ut fra en gjennomført bruksanalyse er det laget en spesifikasjonsliste som danner grunnlag for design kriterier for et nytt innvevd sensor- og styringssystem. Mål for projektet er å designe og implementere et system som tilfredsstiller de definerte brukskriteriene.

Det implementerte systemet inneholder en PID kontroll algoritme og grensesnitt for motor styring; SD minnekort og larging av datalogg; Trådløs Bluetooth grensesnitt, for å overføre datalogg filer og å konfiguere fartøyet; og, sensorer brukt for å operere fartøyet og samle data. Systemet er implementert som et tilpasset kretskort, ulike sensormoduler, programvare/fastvare, og nødvendige mekaniske modifikasjoner. Sensorer som ikke er nødvendig for fartøyets sentrale funksjoner er ikke konfiguert. Sensorene er derimot koblet til og gjort klar for å konfigueres på et senere tidspunkt. En terminal-applikasjon er utviklet som grensesnitt mot fartøyet for datamaskiner med Bluetooth Low Energy.

Fartøyet er testet på land, i ferskvann, og knapt i saltvann. hovedfunksjoner er testet og verifisert; først i omgivelser uten vann, og deretter i et ferskvannsbasseng. Et ferskvann er vurdert passende for å stille inn PID regulatoren, hvor PID koeffisienter er funnet med systemrespons som gjør at fartøyet holder den gitte dybden. Fartøyet er testet i saltvann med koeffisientverdiene funnet i ferskvann, der systemresponsen viser seg å være annerledes og ikke egnet for operasjonell bruk. Videre test er nødvendig for å oppnå brukbar systemrespons for saltvann.

Egenskaper for trådløs luft-vann grensesnitt er undersøkt ved å teste to ulike antennekonfigurasjoner i fersk- og saltvann. Det er funnet at en FlexPIFA antenne gir bedre resultat enn en keramisk chip monopol antenne; der stabil forbindelse er opprettholdt for dybder ned til 10 cm i ferskvann, og 5 cm i saltvann.

Acknowledgement

I would like to thank my supervisors Bjørn B. Larsen at the Department of Electronic Systems and Jo Arve Alfredsen at the Department of Engineering Cybernetics, for theoretical and practical guidance.

Thank you Glenn Angell, at the ITK workshop, for invaluable help with mechanical modifications. Thank you Stefano Brevik Bertelli, Patrick Christian Börsch, and Åsmund Stavdahl for letting me use ITK workshop's equipment. Thank you Tim Cato Netland at the Department of Electronic Systems for letting me perform tests in the acoustics's water tank.

Thank you Mom and Dad for endless support and encouragement. Thank you to immediate and extended family for the safety and support you provide. There are too many names.

Thank you Halvard Berge for help, encouragement, and for providing a physical workspace in extraordinary circumstances.

Thank you Arnout Klinkenberg, Cody J. Miller, and Arley Loewen for friendship, and for the honest and helpful feedback in the writing of this thesis.

Author

Odd Inge Halsos

Contents

Figures

Tables

Abbreviations

BV	Buoyancy Vehicle
PVC	PolyVinyl Chloride
PID	Proportional Integral Derivative
EM	Electromagnetic
MCU	Microcontroller Unit
GPIO	General Purpose Input/Output
ADC	Analog-to-Digital Converter
PWM	Pulse-Width Modulation
RGB	Red, Green, Blue
LED	Light-Emitting Diode
I^2C	Inter-Integrated Circuit
SPI	Serial-Peripheral Interface
UART	Universal Asynchronous Receiver/Transmitter
SoC	System on a Chip
BLE	Bluetooth Low Energy
TWIM	Two-Wire Interface Master
SCK	Serial-clock
MISO	Master-In-Slave-Out
MOSI	Master-Out-Slave-In
CS	Chip-Select
SAADC	Successive Approximation Analog-to-Digital Converter
EasyDMA	Easy-to-use Direct Memory Access
CPU	Central processing Unit
RAM	Random-Access Memory

FlexPIFA™	Flexible Adhesive-Backed Planar Inverted-F Antenna
DSP	Digital Signal Processing
GND	Ground
Op-Amp	Operational Amplifier
LDO	Low Drop-Out
DK	Development Kit
SDK	Software Development Kit
SWD	Serial Wire Debug
SIG	Bluetooth Special Interest Group
ISM	Industrial, Scientific, and Medical
PHY	Physical Layer
PDU	Protocol Data Unit
CRC	Cyclic Redundancy Check
MIC	Message Integrity Check
MAC	Media Access Control
HCI	Host Controller Interface
L2CAP	Logical Link Control and Adaption Layer Protocol
SMP	Security Manager Protocol
ATT	Attribute Protocol
GATT	Generic Attribute Profile
GAP	Generic Access Profile
BOM	Bill of Materials
MTU	Maximum Transmission Unit
NUS	Nordic UART Service
UUID	Universally Unique Identifier
RX	Receive
TX	Transmit
RTC	Real Time Counter
FSM	Finite State Machine
LFCLK	Low Frequency Clock
EMA	Exponential Moving Average

psi	Pound-per-square-inch
pps	pulses/microsteps per second
rps	Rounds Per Second
rpm	Rounds Per Minute
CTD	Conductivity, Temperature, and Depth
RSSI	Received Signal Strength Indicator
kB	kilo Byte
atm	Atmospheric Pressure

Chapter 1

Introduction

Gathering subsurface oceanic data is a challenging and time consuming task. Implementation of embedded systems to control a vessel with sensors and data storage provide a way for autonomous investigation of underwater environments [1].

A Buoyancy Vehicle (BV) is designed for this very task. By utilizing buoyancy effects, caused by water with density as a function of depth(pressure), salinity, and temperature a vehicle capable of adjusting its own volume can adjust its vertical position. By settling at a specific depth it can act as a Lagrangian drifter. The lagrangian drifter is an object, or vehicle, that is deployed such that it is able to give quantitative description of the flow. A typical application is to track the flow of large scale ocean currents, and small local water circulations [2]. The vertical moving buoyancy vehicle, with the ability to settle at any depth in its given range, can be used to track and observe underwater currents.

A project carried out by Stian Børseth [3] and supervised by Jo Arve Alfredsen at the department of Engineering Cybernetics, resulted in a small scale buoyancy vehicle designed for vertical movement at depth from 0 to 50 meters.

1.1 Project Formulation

This project is aiming at continuing Børseth's work, where a re-designed embedded control -and sensor system are to be made. Through a use-case analysis a set of requirements are derived, which serve as design criteria for the system. Figure 1.1 illustrate the system broken down into a few core components. A battery pack, an analog pressure sensor, and a stepper motor configured as a linear actuator [3], is available from the previous project. A new PCB main-board with sensors, data storage, and interfacing to external components is made. Lastly a wireless communication link is set up that connect to a computer with a user interface, to configure the device and transfer acquired sensor data.

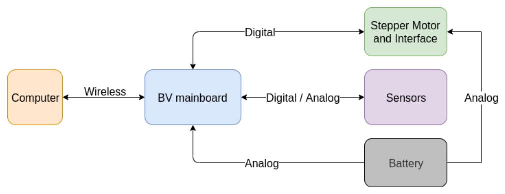


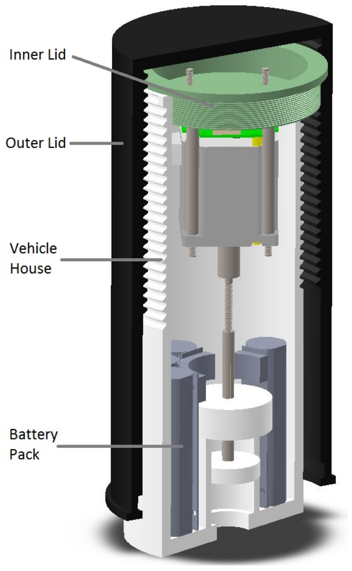
Figure 1.1: High level system diagram shows the mainboard with digital and analog interface to sensors and stepper motor, battery powered mainboard and motor, and wireless communication link to a computer with a user interface for system configuration

1.2 Previous Work

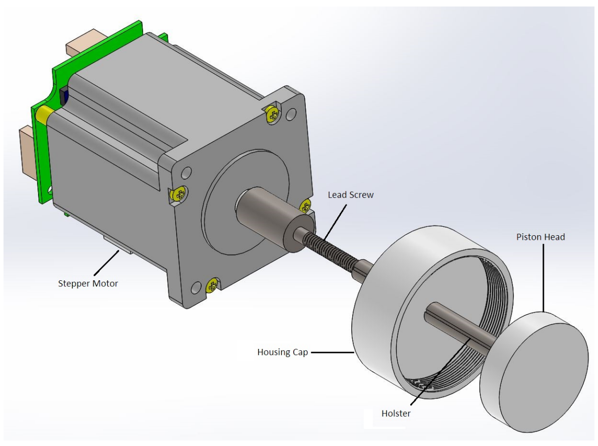
The buoyancy vehicle prototype, designed by Børseth [3], is illustrated in figure 1.2a and 1.2b. It adjusts its volume with the help of an electric linear actuator and a piston. Volume is reduced when the piston head is pulled in (up towards the motor) to let water into the cylindrical piston. This causes the vehicle density to increase and sink until it hits water with equal density. The vehicle has a cylindrical shape, and is realized in rigid polyvinyl chloride (PVC) material, as this is optimal to withstand high pressure.

The vertical position is adjusted and maintained by a simple embedded system based around an Arduino Nano evaluation board. It interfaces to a micro SD card module for data logging, a pressure sensor for depth measurement, battery voltage measurement, and a stepper Motor to drive the piston head. The system is supplied by 16 alkaline batteries, sufficient to provide the nominal stepper motor voltage of 24 V. The vertical position is maintained by a Proportional–Integral–Derivative (PID) control algorithm. The controller receives a target depth and, based on depth samples from a pressure sensor, calculate a new piston position. The new piston position changes the vehicle density, which generates up or down force and vertical movement towards the target depth [3].

The **vehicle house** is considered the mainframe as it holds the necessary components, as well as withstands the pressure of the surrounding water. The **outer lid** is screwed on to the vehicle house, and sealed with an O-ring. The initial volume can be adjusted by the outer lid, in case the water density requires the initial BV density to be adjusted or the vehicle carries additional payload [3]. The **inner lid** function as the mounting point for the stepper motor and the circuit board. The stepper motor is fastened with four screw holders under the lid, and the circuit board is fastened with the same four screw holders on top, each with a locking nut and two washers. A custom **battery pack** was made to fit the 16 alkaline batteries [3].



(a)



(b)

Figure 1.2: a: Buoyancy Vehicle cross section shows the different parts of the buoyancy vehicle prototype. **b:** Shows the linear actuator consisting of; a stepper motor, lead screw attached to a threaded holster, a housing cap screwed, and a piston head. Reproduced with permission from [3].

1.3 Motivation

The buoyancy vehicle designed by Børseth is a lightweight underwater vehicle, which is simple to bring to a test location and make a minimum impact on the environment where it's operated. It has great potential to become an useful tool with many applications in underwater measurement and data acquisition. To bring the project closer to a fully operable state a use-case analysis is carried out where a sensor -and control system is design and implemented to fulfill the specified requirements.

1.4 Project Scope and Goal

The main goal and priority of this project is to design and implement an embedded control- and sensor system to satisfy the use-case requirements presented in section 3.1. Necessary sub-goals include; investigate use-cases, define necessary system requirements for each use-case, implement software and hardware to satisfy requirements, and perform test on land and in water.
As a starting point the existing prototype is reviewed, use cases are defined, and a set of system requirements are derived.

The finished system is planned to be tested on land, thereafter in a freshwater tank, and finally in the ocean.

1.5 Applications

The buoyancy vehicle is designed to be configured through wireless communication, and deployed on a subsurface autonomous data acquisition mission. Its purpose is to acquire data from various sensors, as well as operate as a subsurface drifter, in order to measure underwater currents at a given depth.

1.6 Report Outline

This thesis is outlined for the reader to gain a working understanding of the system, and to document the work carried out in this project.

Chapter 1 - Introduction present a few topics regarding project formulation, motivation, and goals. A problem is formulated, motivation for solving the problem is presented, and project scope and goal is defined, as well as a brief report outline.

Chapter 2 - Background and Theory provide relevant background information and a theoretical foundation.

Chapter 3 - Use Case, Requirements, and Modules present use cases, requirements derived for each use case, and a overview of each module needed to fulfill the requirements.

Chapter 4 - Printed Circuit Board presents a detailed overview of the printed circuit board, with schematic, layout, and the procedure of placing the order at manufacturing company.

Chapter 5 - Firmware present information of relevant components within the embedded software stack. Wireless protocol is explained, and central implementation details are highlighted. The various software modules and drivers are described, and essential aspects related to motor configuration and interface are presented.

Chapter 6 - Mechanical Modifications provide brief presentation of necessary mechanical modifications to the prototype.

Chapter 7 - Test and verification present the test procedures carried out to verify correct functional behavior on land and in water, detect bugs and errors, and tune PID controller.

Chapter 8 - Results and Discussion present and discuss test results to analyze different aspects of the design, provide recommendations for future work, and to draw final conclusions.

Chapter 9 - Conclusion provides a summarizing conclusion, where important results are highlighted, and suggestions for further development.

Chapter 2

Background and Theory

This chapter serves as a presentation of some relevant background theory needed to understand the concepts of which the project is based upon. This chapter covers the following topics; A theoretical motivation for the buoyancy effect, buoyancy in nature, buoyancy vehicle technology, a brief note on embedded systems, presentation of a closed loop system and PID controller, and finally a relatively brief theoretical presentation of conditions and consequence for wireless communication between air and water.

2.1 Bouyancy Effect

This section presents buoyancy effect theory, an example of buoyancy in nature, various technologies for regulating buoyancy in buoyancy vehicles.

2.1.1 Bouyancy Force

An object submerged in fluid, partly or entirely, experiences two vertical forces: gravitational weight and buoyancy force. The gravitational weight exerts a force equal to the objects mass times the magnitude of the gravitational field at the objects location, with the force pointing toward the Earth's center. The buoyancy force points in the opposite direction, away from the Earth's center, and is a result of the objects volume that displaces the submerging fluid. The Archimedes principle states:

"*The magnitude of the buoyant force on an object always equals the weight of the fluid displaced by the object*"[4, p. 424].

$$B = \rho_{fluid} g V_{disp}$$

where $\rho_{fluid} V_{disp}$ is the fluid mass displaced by the object[4]. Thus the total buoyancy force is:

$$B = m_{fluid} g$$

The result is consistent with the statement of Archimedes principles above.

In order to maintain equilibrium, the gravitational weight force and buoyancy force need to be of equal magnitude in the opposite direction.

$$F_G = \int_v \rho_{body} g dV$$

$$F_B = -\oint_S \rho dS = -\int_v \rho_{fluid} \cdot g \cdot dV \tag{2.1}$$

$$F_{total} = F_{body} + F_{fluid} = \int_v (\rho_{body} - \rho_{fluid}) \cdot g \cdot dV \tag{2.2}$$

Changing the objects mass density will change the net force. The object will then move in the direction of the resultant force until equilibrium is obtained. As buoyancy vehicle mass density is dependent on mass and volume

$$\rho = \frac{mass}{Volume} \tag{2.3}$$

Changing the body volume will change the body density [5]. The vehicle will then be able to move up and down in water by change the vehicle volume.

2.1.2 Buoyancy in Nature

The use of buoyancy as a means to adjust and maintain vertical position in water can be observed in salmon and brook trout [6]. The swim-bladder in fish is expanded to reduce its density, and as a result of the buoyancy effect obtains and maintains the desired depth without having to spend excess energy [6].

2.1.3 Buoyancy Vehicle

Similar to the salmon and brook trout described above, a BV is designed to change its density to adjust its vertical position. Several technologies have been developed to achieve this:

Compressed Gas:
The patent from 1946, available in [7], describes a buoyancy vehicle with a container carrying compressed gas upon deployment. The vehicle is quickly descended to a given depth, and is designed to operate with minimum oscillation during operation. In order to maintain depth a valve is adjusted to release gas in just the right amount to keep the vehicle steady.

Mass Removal:
A patent filed in 1993 and published 1995, available in [8], describes a BV operated by changing its mass. The system has two containers where one contains a

heavy liquid and another a light liquid. A computer system monitors current depth by means of a pressure sensor, and controls the vertical movement by releasing one of the two liquids. The heavy liquid is released to reduce mass, which reduces density, creates positive buoyancy, and brings the vehicle up toward the surface. The light liquid is released in order to increase mass, which increases density, and creates negative buoyancy that result in movement downwards.

Hydraulic fluid:
The autonomous Lagrangian circulation explorer (ALACE)[9] is a float that is operated by pumping hydraulic fluid from an internal reservoir to an external bladder. Relocating fluid to the external bladder increases float volume and creates buoyancy.

Phase Transition:
Phase transition[1] can result in change in volume. This has been investigated by H. Yamamoto and K. Shibuya in [10], by experiments on paraffin wax. It is found that the wax did change buoyancy as it was changing form from solid to liquid by means of a heating device. The similar mechanism can be found in the sperm whale, where it controls its buoyancy by melting and coagulating the spermaceti oil in its head by adjusting the temperature by $3°C$[11].

Linear Actuator:
The Mini-Autonomously Underwater Explorer (M-AUE) project, found in [1], is a swarm of 1,5 liter sized buoyancy vehicles aimed at tracking of plankton behavior. The small form factor is made possible by using an electric motor coupled with a vertically moving piston. The motor shaft rotation is in linear correlation to the vertical movement of the piston.

2.2 Embedded Systems

An embedded system is a use-case specific computer. The embedded computer is designed and implemented with hardware and software to fit specifications derived from each use case. The system is often centered around a microcontroller unit (MCU) where a central processing unit (CPU), memory, and additional peripheral hardware is integrated on a single chip[12]. Common constraints of embedded systems are energy consumption (as devices are often powered by batteries), physical size, processor performance, memory and storage capacity, and cost[13]. It is essential that a system is neither over-engineered, where the system is implemented with exceedingly more resources than actually needed, as it will increases its cost, or under engineered, where resources are not fitted proportionally to fulfill its use cases[13].

[1]**Phase Transition is the transition between solid, liquid, and gaseous state of a material.**

2.3 Closed Loop System

In control system theory a closed loop system is one that calculates an output value, as explained in [14]. For instance a new actuator position, based on a desired set point value and a negative feedback loop. The feedback value is often a sensor measuring temperature (°C), pressure (psi/pascal), or any other sensor that measures a value affected by the loop output value. Figure 2.1 shows a representation of a closed loop system, with a set point added to the system loop. The process variable, which is the the system parameter needed to be controlled (temperature, distance, pressure, etc), is subtracted from the set point to find the difference. The compensator, which is a control system algorithm, takes the difference and calculate an actuator output which is given to the system actuator driver. A sensor provides the change in process variable value as feedback for the next calculation. The loop system will gradually pull the process variable to the set point where it reaches *steady state*.

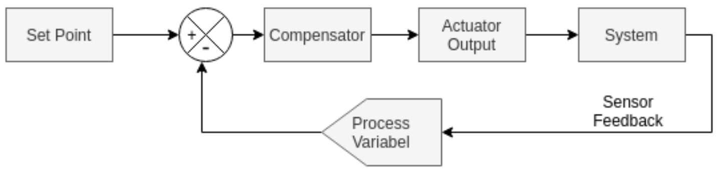


Figure 2.1: A modeled closed loop system diagram with a given set point and measured feedback value creating an error value that is processed in the compensator to create an actuator output to change the system state, resulting in a new feedback value. Adapted from [14].

Figure 2.2 gives an overview of the most important system response characteristics explained in [14]. Rise Time is the time from 10 % to 90 % of steady state, or final value. Percent Overshoot gives a value for how much the steady state value is overshoot. It is expressed in percentage of steady state value. Settling time is the time from when the new set point is given to when the new process variable is within a certain percentage of steady state. It is common to use 5 % of steady state. Steady state error is a measure for the difference between set point and the final steady state value. Finally loop cycle time is the time interval between each calculation cycle. A complex, or a rapid changing system requires short loop cycle time, a slower changing system won't need new calculations as often.

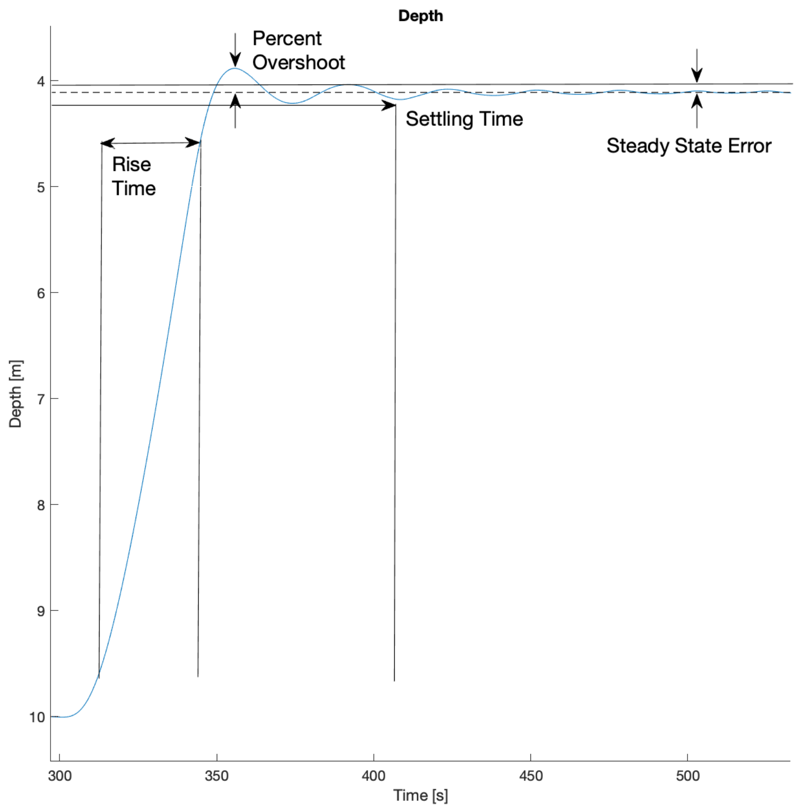


Figure 2.2: Illustration of a characteristic PID system response with Rise Time, Percent Overshoot, Settling Time, and Steady State Error exemplified. Adapted from [14].

2.4 PID Controller

PID controller, illustrated in Figure 2.3, is one of the most commonly used control algorithms in industrial applications where rotational and/or linear actuators are utilized in positioning systems[15]. PID is an acronym for Proportional-Integral-Derivative that represent the three coefficients K_p, K_i, K_d, which are tuned to find the optimal system response[15].

The mathematical representation of the algorithm is shown in the following two equations[16]:

$$u(t) = K_P e(t) + K_i \int_0^t e(t')det' + K_d \frac{de(t)}{dt}$$

$$u(t) = K_P e(t) + \frac{1}{T_i} \int_0^t e(t')det' + T_d \frac{de(t)}{dt}$$

Though they are both the same, the first equation is populated with the coefficients, making it easy to see how they directly affect the output value u(t). The equation below serves as a more direct representation, where the K_i, K_d coefficients are substituted with $\frac{1}{T_i}$ and T_d respectively. Here T represents the time period for which the given term is applied. As a simple explanation, provided by [16], the first product is the error signal multiplied with the proportional gain coefficient, the second product is the integral coefficient multiplied by the integral (or sum) of the error signal at a certain frequency (hence $\frac{1}{T_i}$), and the last product is the derivative (rate of change) over a certain time period.

A better understanding of the characteristics of each coefficient is given by [14]:

- **Proportional response:** The proportional component depends on the difference between current process variable value and set point. The difference is defined as the error. K_p is the proportional gain that is multiplied with the error value in order to counteract the error. The proportional gain makes the regulation faster. Large K_p value may lead to oscillation, where a further increased K_p causes oscillation to become unstable and out of control.

- **Integral Response:** The integral component, K_i, sum the error signal, usually in small values at the time. Though the error may be small the integral summation will result in a significant counter error signal. Consequently the error value is pulled towards set point even though the K_p proportional contribution is small (when close to set point). The integral may cause *integral windup* where the output signal is saturated without regulating the process variable towards set point, which in case would bring the error signal towards zero.

- **Derivative response:** The derivative component, K_d, has a dampening effect by decreasing the output value in case of a rapid increase. When the derivative response is proportional to the process variables rate of change, an increase of the K_d coefficient will cause the system to have a stronger reaction to the changing process variable and the error value. It will therefore increase the speed of the overall system response. As the derivative response is sensitive to noise, the derivative coefficient is often small in the case where feedback signal is noisy. The reason is that too large derivative term can make the control system unstable.

PID Tuning: Once an intuitive understanding of the PID controller is gained, and the significance of K_p, K_i, and K_d are known, the controller can be tuned by trial and error[14]. A common method, according to [14], is to set the K_i and K_d terms to zero, and increase K_p until the loop output oscillates. Then, When K_p is found with a satisfiable fast response, the integral term is increased to break the oscillating behavior and reduce the steady state error. The integral will create overshoot, however this is necessary for a fast system response. Finally the K_d term is tuned for a quick settling time. The K_d term provides a more responsive system, but will, however, cause the system to be susceptible to noise.

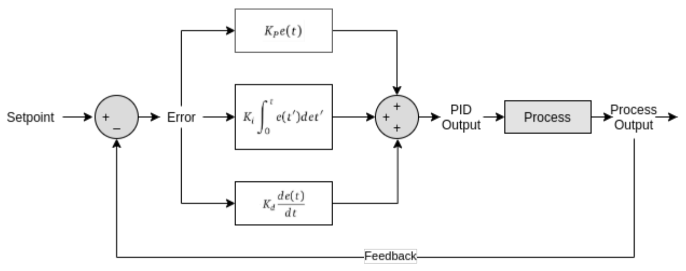


Figure 2.3: Block diagram illustration of a typical PID controller. Adapted from [15].

2.5 Air-to-Water Wireless Communication

An active radio link, between vehicle and a user device, is needed to transfer data files, configuration data, and system commands. In configuration mode the vehicle is designed to float precisely at, or just below, the surface. This is a challenging environment for wireless transmission, as the wireless path includes several different obstacles, including: user device cover material, free air, water, rigid PVC, other objects, and interference with other ISM band frequencies.

High frequency electromagnetic (EM) waves have, until recent years, been thought of as unsuitable for under water communication. Research has since suggested that frequencies in the 2,4 GHz ISM band may be suitable for high data rate communication in short distances of up to 17 cm in fresh water[17].

The path loss of wireless communication in the 2,4 GHz ISM band between user device and buoyancy vehicle include too many variables and uncertainties for a full theoretical outline. A brief outline of some challenges related to the air-water radio link is presented.

The following parameters are used to calculate the absorption constant of water:

- $\omega = 2\pi f$ with frequency f: 2,4 GHz
- Magnetic Permeability of free space μ_0: $4\pi \cdot 10^{-7}$
- Magnetic Permeability of the Medium μ_r: $4\pi \cdot 10^{-7}$ Both air and water are non-magnetic, permeability of water does not affect EM propagation[18].
- Dielectric Permittivity of free space ϵ_0: $8,85 \cdot 10^{-12}$
- Dielectric Permittivity of the Medium ϵ_r: The abillity of a medium to conduct electric field. For both fresh water and sea water assuming $\epsilon_r \approx 81$[18].
- Electrical Conductivity of the Medium σ: Freshwater: 0,01, seawater: 4. Conductivity depends on the number of ions present in the medium. When EM waves propagate through the medium, water, the wave is reflected.

Fresh water has a conductivity of 0,01 $\frac{S}{m}$, and seawater is commonly given as 4 $\frac{S}{m}$. Conductivity of seawater is 400 times higher than freshwater, which has a significant effect on attenuation. The conductivity of water has direct relationship to the propagation and attenuation of EM waves. With increased conductivity the attenuation increase, and consequently the distance waves travel reduced.[18]

To provide insight into some effects related to electromagnetic waves traveling from air to water, the calculation of transmission loss in air-water penetration is outlined as it is presented in [19]:

The incident power is:

$$P_i = Re\left\{\frac{|E_i|^2}{2\eta_0}\right\}$$

where E_i is incident electric field, and η_0 is intrinsic impedance of air given by:

$$\eta_0 = \sqrt{\frac{\mu_0}{\epsilon_0}}$$

The transmitted power in water is:

$$P_t = Re\left\{\frac{|E_t|^2}{2\eta_1}\right\}$$

where E_t is transmitted electric field, and η_1 is the intrinsic impedance of water given by:

$$\eta_1 = \sqrt{\frac{\mu_0}{\epsilon_0\bar{\epsilon_r}}}$$

$\bar{\epsilon_r}$ is the relative complex permittivity of fresh water.

As the transmitted power P_t also can be expressed as:

$$P_t = Re\left\{\frac{|E_t|^2}{2\eta_1}\right\}|T|^2\, e^{-2\alpha d}$$

where $|T|$ is the transition coefficient for electric field transitioning from air to water as the relation of :

$$T = \frac{E_T}{E_i} = \frac{2\eta_1}{\eta_0+\eta_1}$$

Now the transmission loss coefficient α_t, the power loss caused by air-water interface, can be calculated in dB:

$$\alpha_t = 10log_{10}\left(|T|^2 Re\left(\frac{\eta_0}{\eta_1^*}\right)\right)$$

The propagation loss in water α is given by:

$$\alpha = \omega\sqrt{\mu\epsilon}\left\{\frac{1}{2}\left[\sqrt{1+\left(\frac{\sigma}{\omega\epsilon}\right)^2}-1\right]\right\}^{\frac{1}{2}}$$

and propagation loss in dB is calculated by:

$$\alpha_p = 10log_{10}\left(e^{-2\alpha d}\right)$$

Where d is the depth, or distance, traveled in water.

Finally the total loss of power when traveling from air to water, and a certain distance through water, upon penetrating the surface at normal is:

$$\alpha_{total} = \alpha_t + \alpha_p = 10log\left(\frac{P_t}{P_i}\right)$$

In case the air-water transition take place at an angle other than normal, further calculations are necessary to account for the refraction effects. In short, the transmission coefficient is decomposed in parallel and normal components, Snell's law is used to find the intrinsic and transmission angle, which is then used to calculate the transmission coefficient values[19]. Transmission loss and propagation loss coefficients α_t and α are then calculated for the parallel and normal component[19]. These are finally combined to give the total loss coefficients in parallel and normal direction[19].

The vehicle's application areas are also in seawater. [17] and [18] provide an abstract view of the effect that seawater has wave propagation compared to freshwater. The absorption constant of freshwater is approximated by equation 2 in [17]:

$$\alpha_{freshwater} \approx \frac{\sigma}{2}\sqrt{\frac{\mu_r \cdot \mu_0}{(1+\epsilon_r)\cdot\epsilon_0}}$$

The absorption constant of seawater is approximated by equation 8 in [18]:

$$\alpha_{seahwater} \approx \sqrt{f\mu_r\sigma\pi}$$

The equations above indicate a higher degree of absorption of EM waves in seawater than freshwater.

Chapter 3

Use Case, Specifications, and Modules

This chapter presents the use case analysis with the obtained use cases, system specifications, and modules. Some additional theory is provided in subsection 3.3.9 related to measuring battery and pressure.

3.1 Use Cases

In order to come up with relevant design criteria a set of clearly defined use cases are made. Each use case is derived by analyzing the vehicles purpose, by isolating each step from off-mode, through a completed mission, and back into off-mode. During the analysis it is important to make an effort to think through risk factors in stages, such as initialization, configuration, and mission to avoid or be able to handle errors that may occur. This will hopefully avoid failure that, in worst case, may lead to losing the vehicle under water.

Each use case is briefly described, while detailed definitions are provided in Appendix A.

- **Power-On:** The user power up the device with a magnetic activated button. The Default state after power up is standby/Idle.

- **Configuration:** The vehicle is configured through wireless communication, where a number of variables such as depth, time, etc. is given for each mission in the next dive.

- **Mission Deployment:** The vehicle is placed in water (if not already floating in water) and start the dive. during a dive each mission has a defined depth and time.

- **Pick-Up:** The mission is completed, or an error has occurred, in which case the dive and mission is aborted. In the event of errors an exception state is entered, which make sure the vehicle float to the surface. The vehicle has floated to the surface to transfer data and to be re-configured. The vehicle can either be picked up from the water, or remain for a quick turn-around where data is transferred before the next dive.

- **Power-Off:** The vehicle is picked up from the water and powered down by a magnetic activated button.

3.2 System Specifications

From the use case analysis above a set of system requirements is derived and from these a final list of system specifications are made. The specifications define guidelines for the following design process. The full list is given in Appendix B.

The summarized list of specifications is included here for convenience:

Specification Summary:

- The vehicle should have a magnetic activated mode-button that powers the system on, changes mode, and power the system off. The system is powered off by entering Idle mode, where a timer power the system down after an inactive period.

- A magnetic activated mode-button should be implemented by reed-relay or a Hall effect sensor.

- The system need modules that satisfy the following requirements: Antenna, sensors, and indication LED visible on the outside of the vehicle.

- The vehicle should have several operation modes: Standby/Idle (power save), Config (configuration mode), Deploy (Mission deployment), Pick-Up (Pick up from water and/or: transfer data, re-configure, and re-deploy), error mode, and low-power-pickup-mode.

- A simple configuration interface run on a configuration computer should be designed to communicate with the vehicle.

- The vehicle should send acknowledge message back to the user device when configuration is received.

- Battery voltage should be measured and logged.

- End limit switches should be installed in order to avoid damages to piston and linear actuator in case motor fail to stop in time.

- For future expansion there need to be an available serial port connection in case external sensors are added.
- As the vehicle is deployed in a marine environment with limited battery life the system should be designed for low power.
- The vehicle is designed for vertical movement between 0 meter and 50 meters below the surface.

Test:

- A functional test on land to make sure the system behave in line with specifications.
- The vehicle should be tested in a water tank with no current flow or in the ocean with minimum current flow to test functionality and tune the PID controller.
- The vehicle should be tested in a large-scale sea cage to measure water-current flow.

3.3 Modules

Based on each use case and the resulting list of specifications the embedded computer is equipped with a number of modules. Figure 3.1 gives an overview of how the main components are interfaced. Each module is in breadboard friendly size to set up the test and development environment. Each module can then be tested, individually and as a working system, before a Printed Circuit Board (PCB) is designed and produced. Though modules will take up more space, the benefit is that modules can be tested with software as soon as they arrive. Otherwise most components must wait for revision 1 of the PCB to be designed and assembled. Moreover, the revision 1 PCB often have design flaws that can cause further delay to component testing and software design.

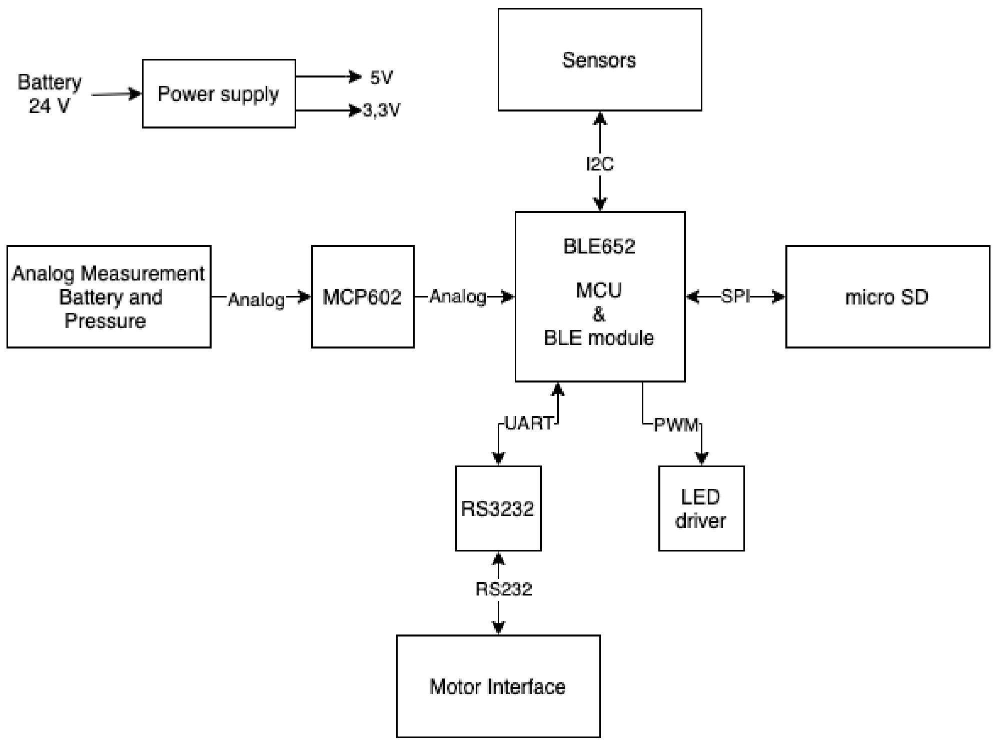


Figure 3.1: High level system diagram to show how modules are grouped and interfaced.

The module selection is a recursive process. Different components are evaluated as to whether they fit individually and as a system. Factors such as current consumption and peripheral interfacing to the MCU are essential topics to evaluate.

The MCU is selected based on several criteria. The total number of available General Purpose Input/Output (GPIO) pins, number of analog-to-digital (ADC) pins, available serial communication peripherals, wireless compatibility, total current consumption, etc It also need sufficient storage capacity for program code, RAM and flash memory.

Table 3.1: List of modules with the total number of digital/analog pins and Serial Protocols required in the MCU.

	Digital IN/OUT	Analog IN/OUT	Peripheral
RGB LED and PicoBick v12 Driver	3		
Hall-Effect Sensor	1		I2C
TMP 117 Sparkfun	1		I2C
9-Axis Direction	1		I2C
Battery Measurement			ADC/MCP602
PX3 Pressure			ADC/MCP602
MCP602 Operational Amplifier		2	
SD-module			SPI
Motor Interface			RS232
RS3232			UART
2x Limit switches			Motor Interface
Serial Com.For extended sensors			I^2C
MCU module require:	6	2	I^2C, SPI, UART
Including:	$I^2C = 2$, SPI = 4,	UART = 2	

Table 3.1 provide the final list of modules and how they interface to the MCU. In total 16 GPIO pins are required. The 6 digital pins are 3 Pulse-Width Modulation (PWM) signals, to adjust RGB LED brightness, and 3 digital input pins for interrupt signals from modules. Two pins are analog signals, sampled by the ADC peripheral. 8 pins are needed for serial peripherals: I^2C (2), SPI (4), and UART (2).

3.3.1 nRF52832 System on a Chip

The system is centered around the nRF52832-QFAA System on a Chip (SoC) from Nordic Semiconductor. This 32-bit SoC features the appropriate number of GPIO pins and peripherals, including Bluetooth Low Energy (BLE) radio hardware.

Peripherals

Peripherals are hardware devices on chip that enable it to interface with external components and devices. It is essential that the SoC have the required peripherals to interface all the necessary modules in the system.

- General Purpose Input/Output **GPIO** peripheral are physical pins that extend the SoC functionality to be able the interface with external components through input signals, output signals, or both. Each individual GPIO can be configured with various parameters, such as drive direction, drive strength, internal pull-up or pull-down, etc.[20, p. 111 - 154]

- Two-Wire-Interface Master **TWIM** is a two wire half-duplex bus that can communicate with multiple slave devices on the same bus. It is I^2C-compatible and can interface 127 individual addressable device on a single bus.[20, p. 305 - 317]

- Serial-Peripheral-Interface **SPI** is a 4 pin, Serial-clock (SCK), Master-In-Slave-Out (MISO), Master-Out-Slave-In (MOSI), and Chip-Select (CS). Each individual slave device share the 3 first lines and have its CS pin tied to an separate GPIO pin on the master device. Each device which CS pin is pulled low is written to simultaneously. Care must be taken that CS pin is pulled low for one device only when a slave is to be read, otherwise multiple slaves will send competing data on the MISO line, which cause corrupted messages.[20, p. 513 - 519]

- Universal-Asynchronous-Receive/Transmit **UART** is a full-duplex, asynchronous serial communication peripheral with baudrate up to 1 Mbps. It is implemented with a 6 bytes First-In-First-Out (FIFO) buffer in the receiver chain, which make reception faster and more reliable with less risk over overwriting data.[20, p. 531 - 539]

- Pulse-With modulation **PWM** utilize a counter to drive assigned GPIO pins high or low at given intervals. The period is configured for each PWM instance and a counter value is set for up to 3 channels per instance. When the counter reach the preset counter value for a given channel the assigned GPIO pin output state is inverted. Typical application is to adjust LED brightness or control servo motors.[20, p. 495 - 512]

- The Successive Approximation Analog-to-Digital Converter **SAADC** samples analog input signals to generate a digital representation. 8-bit, 10-bit, or 12-bit samples can be generated by direct sampling, or 14-bit with oversampling. In this project the SAADC utilize two separate channels that samples battery voltage and pressure sensor data voltage. For each channel there are one specific GPIO pin assigned as AIN[channel].[20, p. 357 - 391]

- An important peripheral device is the Easy-to-use Direct Memory Access **(EasyDMA)**. EasyDMA is an Advanced High-performance Bus-master, similar to the CPU. It allow for direct data transfer between peripheral and RAM (Random Access Memory) without interrupting the CPU. This is a faster way of transferring data, and power is saved, and less time is spent by the CPU.[20, p. 27 - 28]

3.3.2 Laird BL652 Radio Module

The nRF52832 chip is mounted on the Laird BL652 Radio module[21]. The convenience of modules, mentioned in section 3.3, become evident as the need for soldering SoC related components, antenna design, and potential hardware issues with the necessary debugging is avoided.

Using a module is a trade-off between time, cost, and available GPIO/peripheral pins. The module can be programmed using the Laird SMART BASIC language, a simplified version of the *Basic* language, to provide a higher abstraction level to programming the device [21]. However, as the high-level language abstract away the low level hardware control, the C language is used, as it provide full access to the hardware. The module has 31 GPIO pins, one less than the available 32 on chip.

Antenna Configuration

The BL652 module come in two different antenna configurations: A ceramic chip monopole antenna[22] and MHF4 connector for external antenna[21], figure 3.2 left and right respectively. Two PCBs are assembled with different antennas; one with chip antenna, and one with a FlexPIFA[23] (Flexible Adhesive-Backed Planar Inverted-F Antenna) external antenna, see Figure 3.3.

The two antennas each have benefits and drawbacks:

- The Chip antenna is already soldered on to the PCB and no further considerations are needed for antenna placements, except to make sure the PCB has a keep-out area to avoid disturbing the wireless signal. Peak gain is limited to a typical value of 0,5 dBi[22].

- The FlexPIFA antenna has a peak current of 2,0 dBi[23]. The FlexPIFA antenna therefor has potential for better wireless performance. The downside is that an extra cable and the flexible antenna placement introduce the risk of disturbance of the wireless signal, as well as taking up space in a relatively compact system. The FlexPIFA is a flexible antenna designed to be adhesive mounted to materials such as PVC[23].

The wireless performance of the antenna configurations are tested and compared in free air, freshwater and saltwater.

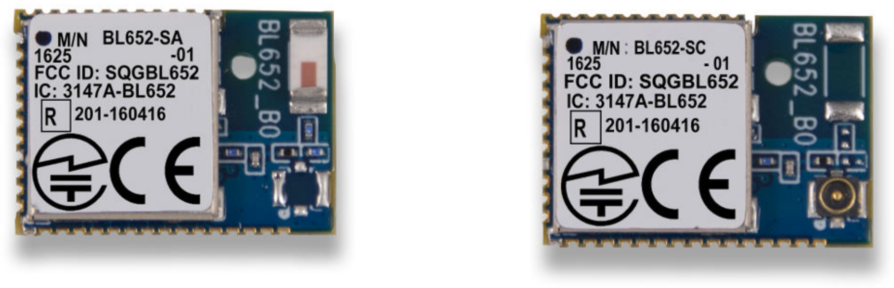


Figure 3.2: BL652 Radio module with two antenna configurations; **Left:** Ceramic chip monopole antenna, **Right:** MHF4 connector. Reproduced from datasheet [21]

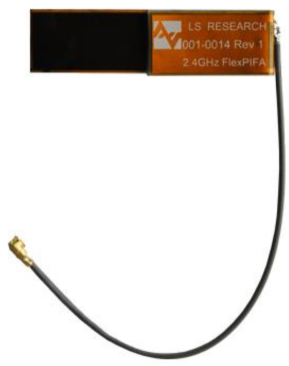


Figure 3.3: FlexPIFA antenna with MHF4L connector designed to be adhesive mounted to material such as PVC. Reproduced from datasheet [22].

3.3.3 Si7210 - Hall Effect Sensor

The Si7210 Hall effect sensors, shown as block diagram in Figure 3.4, from silicon labs[24] provide a chopper-stabilized Hall element with low-noise analog amplifier, a 13-bit analog-to-digital converter, and a I^2C interface for configuration and reading sensor data. A digital signal processing block is also incorporated for precise temperature and offset drift.

The current consumption is down to 50 nA in sleep mode, and An ALERT output pin can be triggered, either high or low, when it passes a set threshold[24].

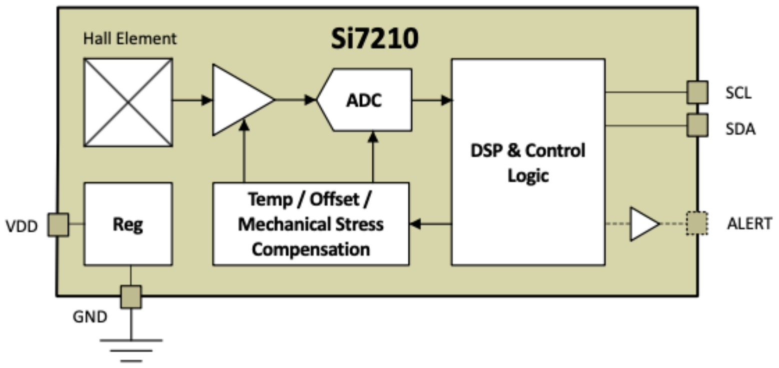


Figure 3.4: Si7210 Block Diagram: Hall Element, Low power analog amplifier, ADC, DSP and control logic, I^2C, and ALERT output pin. Reproduced from datasheet [24].

3.3.4 TMP117 - Temperature Sensor

TMP117 is a Low power, 16-bit resolution, digital temperature sensor from Texas Instruments[25]. It has ±0,1 °C accuracy across the interval -20 °C to 50 °C with 0.0078 °C resolution. The sensor current consumption is 3,5 μA in active mode, and 150 nA in shutdown mode which. The low power consumption minimize self heating effect on measurement accuracy. The sensor has an I^{2}C compatible interface. There are several configuration options, such as temperature offset, and alert function at a set temperature threshold that can trigger an INT output pin. The sensor is mounted on a 2,54x2,54 mm PCB module made by SparkFun Electronics[26].

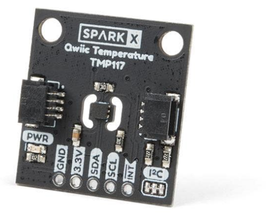


Figure 3.5: TMP117 temperature sensor mounted on a PCB module. Reproduced from [26].

3.3.5 ICM20948 - 9-axis Motion Sensor

The ICM20948 9-axis motion sensor[27] consists of two separate dies within a single 3x3x1 mm package. One die with a 3-axis accelerometer, 3-axis gyroscope, and a Digital Motion Proccessor^{TM} (DMP). The other die has the AK09916 3-axis compass from Asahi Kasei Microdevices Corporation. It also features a 16-bit ADC, programmable digital filters, and internal chip temperature sensor. Figure 3.6 shows the ICM20948, mounted on the PIM448 PCB module. The chip include both I^2C and SPI interface, however the module is limited to I^2C.

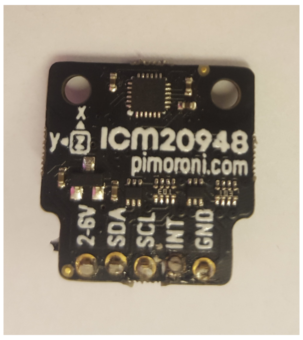


Figure 3.6: Picture taken by the ICM20948 9-axis motion sensor, mounted on the PIM448 module.

3.3.6 PX3 - Pressure Sensor

The Honeywell PX3 - series Heavy Duty Pressure Transducer uses a piezo resistive element with signal conditioning in an application Specific Integrated Circuit (ASIC)[28]. The sensor's specific type number is: PX3AN1BH100PSAAX. It is a sealed gauge sensor with 100 psi pressure range from 14,7 psi to 114,7 psi, where the analog signal output voltage ranges from 0,5 V to 4,5 V, see Figure 3.7[28]. The sensor is calibrated and temperature compensated for the interval -40 °C to 125 °C[28].

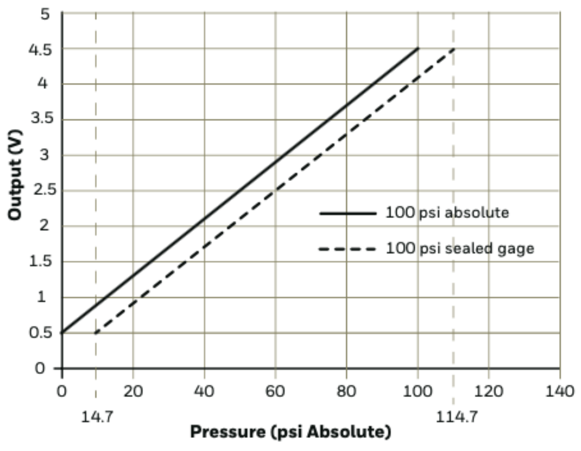


Figure 3.7: PX3 pressure sensor sealed gauged range from 14,7 to 114,7 psi, output voltage: 0,5 V to 4,5 V. Reproduced from datasheet [28].

3.3.7 RGB Power LED and LED Driver

The high Red Green Blue (RGB) power LED has a maximum power consumption of 3,6 W. It is mounted on an aluminum PCB that helps with heat dissipation[29]. It is advised not to look directly at the LED when operated with high brightness.

The PicoBuck v12 LED driver is a three channel, high power, and constant current LED driver module[30]. The module can source up to 330 mA current on each channel, and can be doubled to 660 mA by soldering a jumper. Each channel is dimmed an analog or a PWM signal[30].

(a)

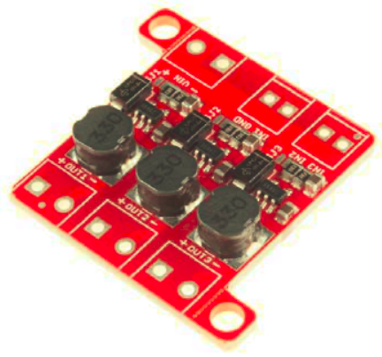

(b)

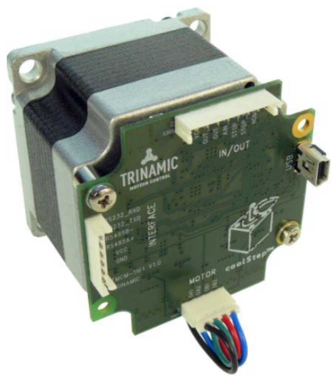
Figure 3.8: a: RGB Power LED. Reproduced from datasheet [29]. **b:** PicoBuck 3 channel LED driver. Reproduced from datasheet [30].

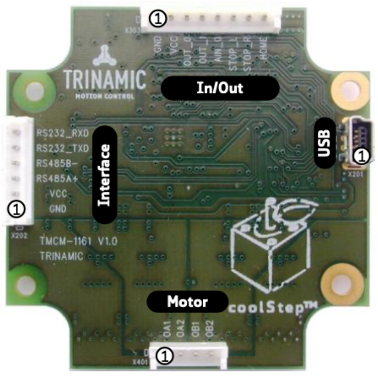
3.3.8 Stepper Motor and Linear Actuator

This section presents and explains the parts related to the stepper motor and linear actuator, which is driving the piston head to change the vehicles volume.

Stepper motor and Controller interface

Figure 3.9a and b shows the Trinamic PD60-3-1161 stepper motor and TMCL-1161 control module. The stepper motor voltage range is 10 V to 30 V, nominal supply voltage of 24 V[31]. The control module can be interfaced with serial line communication such as USB and RS232, or by dedicated I/O pins[31]. The control module is itself an advanced computer system. It has a microprocessor with flash storage that is programmable with TMCL^{TM} firmware[32]. The motor can be programmed and operated in standalone mode where a program is uploaded and executed on the control module. It can also be operated in direct mode where individual commands are sent over serial communication[32].
Direct mode is chosen, as nRF52832 is the main processing unit, interfaced with RS232 serial communication protocol. The 3,3 V powered RS3232 module is interfaced to nRF52832 with the UART peripheral.


(a)



(b)

Figure 3.9: **a:** PD60-3-1161 Stepper Motor **b:** TMCL-1161 Control Module. Reproduce from datasheet [31].

Linear Actuator

Figure 1.2b illustrate the linear actuator assembled with piston housing cap and piston head. The linear actuator is one of the essential core components adopted from Børseth's work[3]. The stepper is the actuator that create torque which is translated into linear movement. The linear movement is generated through a 1 mm pitched lead screw, connected to the rotating motor shaft, and a threaded holster is screwed on to the lead screw. For each full motor shaft (and lead screw) rotation the holster move 1 mm. Left rotation move the holster up, right rotation move the holster down. The holster has three tracks on its longitude axis that fits through the top of the piston housing cap. The piston cylinder is fastened from the bottom so that the holster and prevent from rotating. The piston head is mounted at the bottom end of the holster, which now is moving up and down depending on the rotational direction of the motor.

End Limit Switches

Limit switches are needed to avoid damage to the linear actuator and piston, in case the motor fail to come to a stop. The motor may cause damage to threads in the linear actuator, or force the piston head to exceed the physical limits of the piston. The control module have 3 I/O pins that can be used as end limit switch input pins, that halt the motor when triggered[32]. One pin for left switch, one for right switch, and one for an optional home switch that can be used for initialization.

The limit switches chosen are 653-D2MQ-4L-1-L switches[33] mounted on 3D printed holders presented in section 6.2. The switch's COM pin is connected to the controller module limit switch input pin, and NC (Normally Closed) is connected to GND[33]. If the signal is left floating an internal pull-up resistors in the controller module trigger a signal to halt the motor[32]. When the switch is triggered (opened), or one of the wires a disconnected, the module input pin is left floating. The motor is then unable to move in that direction[32].

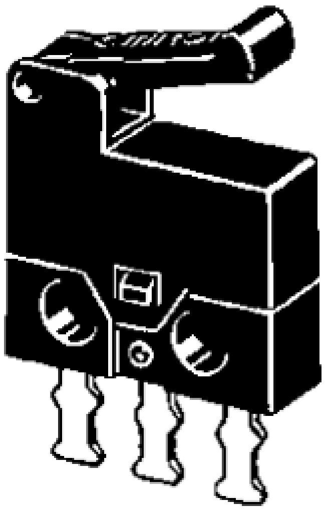


Figure 3.10: 653-D2MQ-4L-1-L End Limit Switch. Reproduced from datasheet [33].

3.3.9 Voltage Measurement: Battery and Pressure

To measure battery and pressure with SAADC the voltage must be reduced to a maximum voltage VDD$_{MCU}$ = 3,3 V. Voltage dividers are designed to downscale the 24 V battery and 4,5 V pressure sensor output accordingly. An MCP602 operational amplifier is configured as a unity-gain buffer[34] to provide impedance matching between SAADC input and measured voltage source. Although MCP602 can be operated at 3,3 V, the supply voltage must be such that common mode input voltage range is VSS-0,3 V to VDD$_{op-amp}$-1,2 V, where VSS is GND[34]. the voltage dividers are designed to give a maximal measured voltage of 3,0 V. Thus the minimum supply voltage for MCP602 is: VDD$_{MCP602}$ > 3,0 V + 1,2 V = 4,2 V.

The voltage dividers are designed with the voltage divider equation, where resistors R1 and R2 are chosen to regulate input voltage V_{in} down to an output voltage V_{out} 3,0 V:

$$V_{out} = V_{in} \cdot \frac{R2}{R1 + R2}$$

Table 3.2: Voltage divider resistor values, and the resulting maximum continuous current flow through the resistors.

Measured Voltage	R1	R2	Max continuous current (I_{const_max})
Battery	69,8 KΩ	10,0 KΩ	300,8 μA
PX3 Pressure Voltage	5,0 KΩ	10,0 KΩ	300,0 μA

With the system in off-mode the voltage dividers continue to have current flowing through them. the current through the dividers should therefor be as low as possible. A small current is, however, susceptible to noise. A trade-off between minimal current drain and a noise resistant signal lead to the voltage dividers in Table 3.2, with the resulting maximal continuous current I_{const_max} calculated from:.

$$I_{const\_max} = \frac{V_{in}}{R1 + R2}$$

Table 3.2 show the resulting voltage divider values to measure battery and pressure, respectively. And the current that flow through the resistors at maximum voltage.

3.3.10 Power Supply

The system is powered by 16 1,5 V alkaline batteries[35] which provides a total supply voltage of 24 V when fully charged. As the battery pack discharges the voltage follows a predictable voltage curve, as seen in example from the datasheet in Figure 3.11[35]. In the example a portable stereo draw 400 mA for 2 hours per day. The total amount of hours from fully charged to empty is 14 hours. A single battery is considered empty at 0,8 V. This adds up to 12,8 V for the 16 batteries. The Stepper motor, control module, and LED drivers are all directly supplied from the unregulated battery voltage[35].

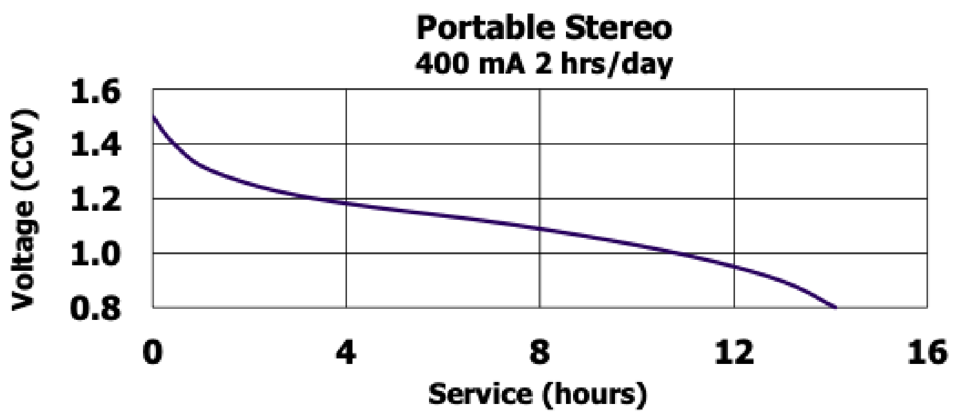


Figure 3.11: Example from data sheet to illustrate the battery discharge curve: 400 mA is drawn for 2 hours per day, total service hours is 14 hours from fully charged to fully discharged. Reproduced from [35].

Table 3.4: System component current specifications

Component	Current (Worst Case)	Current Passive/Sleep
TMP117	3,50 μA	150,00 nA
PX3 pressure	1,63 mA	1,63 mA
MCP602	260 μA	260 μA
9-Axis Direction	3,11 mA Peak	8,00 μA
SD-Card module	100,00 mA	0 A
Stepper motor	4,00 A max peak	0 A
Motor Interface	N/A	N/A
Limit Switches	N/A	N/A
Voltage measurement	600,00 μA	600,80 μA
RGB LED	990,00 mA	0 A
BL652	7,50 mA	400,00 nA
Hall Effect	8,50 mA	400,00 nA

	Current (Worst Case)	Current Passive/Sleep
Summary	5 111,58 mA	2,55 mA

Table 3.3 shows the the available voltage ranges for each module. The stepper motor, motor controller, PicoBuck v12 LED driver is powered directly from battery, the rest need to be regulated to lower voltages. The PX3 pressure sensor and MCP602 Op-Amp require 5 V, while the rest of the system operated at 3,3 V.

Table 3.4 shows the estimated worst case and best case current consumption by each module. Motors RMS current consumption is 2,8 A but it can draw as much as 4 A in short bursts. It is also important to note that this is when motor is configured to draw maximum power[32].

Table 3.3: System Component Voltage Specifications

Component	Voltage min	Voltage Max
TMP117	1,80 V	5,50 V
PX3 pressure	4,75 V	5,25 V
MCP	4,2 V	6,0 V
9-Axis Direction	1,94 V	3,60 V
SD-Card module	2,70 V	3,60 V
Stepper motor	10,00 V (24 V Nominal)	30 V (24 V Nominal)
Motor Interface	N/A	N/A
Limit Switches	N/A	N/A
Voltage measurement	1.60 V	3,00 V
PicoBuck v12	6,0 V	36,0 V
RGB LED	2,2 V	3,8 V
BL652	1,70 V	3,60 V
Hall Effect	1.71 V	5.50 V

	Low Voltage	High Voltage
Summary	3,30V and 5,00 V	24,00 V

A two step configuration is required, where both regulators need to handle at least 121,58 mA. Traco TSR 1-2450 switching regulator[36] is regulating battery voltage down to 5 V, and a LM1117 Low Drop-Out (LDO)[37] regulator is regulating 5 V to 3.3 V. The switching regulator generate an output ripple voltage[36], while the LDO regulator has a cleaner output and is suitable as a second stage, since it will eliminate the switching regulator ripples.

3.4 Software Development Environment

The software development involves both software and hardware tools. The development environment used is SEGGER Embedded Studio, nRF52 Development Kit (DK) is used as prototyping board for breadboard development, and later as programmer for programming the BL652 modules. Nordic Software Development Kit (SDK) is used develop module drivers and access Bluetooth drivers. The Nordic SDK can be downloaded from [38]. The SDK is provided as a resource with examples and ready-made drivers that can be further developed to suite the developers unique application. The nRF52 DK is a development board where all the pins are made easily available to connect to modules on the breadboard. It also include a SEGGER programmer and debugger which can be used to program and debug the development kit as well as external SoC with two-pin Serial Wire Debug (SWD). As the development environment and debugger are from the same developer, debugging is well developed into the IDE.

3.5 Wireless Communication: Bluetooth Low Energy

Bluetooth Low Energy (BLE) is the chosen protocol to create a wireless communication interface between the vehicle and portable devices, such as a mobile phone or tablet, or a dekstop computer. The Bluetooth protocol specifications are proposed and maintained by the Bluetooth Special Interest Group (SIG). Bluetooth is divided in two main branches with different design goals; Bluetooth Classic: Designed for optimized data transmission rate, and BLE: Optimized for low power consumption.[39] Bluetooth operate in the 2,4 GHz ISM band[40].

3.5.1 Bluetooth Low energy vs Bluetooth Classic

Bluetooth was first introduced in 1999 with specification now commonly called Bluetooth Classic, then released as Bluetooth v1.0. In 2010 the Low Energy feature was introduced in v4.0, then introduced as Bluetooth Smart and later commonly known as Bluetooth Low Energy (BLE). Since then Bluetooth classic and BLE has co-existed and been developed in parallel to each other
Bluetooth Classic is intended for short range high data rate transmission, such as audio transmission in the car, or Bluetooth handsfree headsets.[40]

BLE on the other hand is intended for Low data rates, low power consumption, and a wide application span.

Despite being a relatively recently introduced standard, BLE is has a rapid adoption rate due to timely design goals, such as low energy consumption, low cost, low bandwidth, and is the most widespread wireless protocols used for interfacing portable devices[41].

3.5.2 Bluetooth Protocol Architecture

Following is a brief description of the BLE architecture following a bottom up approach where the first three layers are Controller layers that are close to the physical hardware, and the rest are Host layers. These are layers "hidden" underneath the application layers, which is the code developed to create the custom application utilizing the BLE protocol[40].

1. **Physical Layer (PHY):** The modulation/demodulation is handled by the physical layer. Digital data is translated to analog data sent over the antenna. The spectrum is divided in 40 channels over a 2 MHz spacing ranging from 2,4000 GHz to 2,4835 GHz. Channel 37, 38, and 39 are advertising channels. where advertisement packets are broadcastet on all three channels during an advertisement interval. The rest of the channels are data channels used for bi-directional data communication. A chip can have up to three physical layers, 1 MBYPS, 2 MBPS, and CODED, where the nRF52832 chip have 1 MBPS and 2 MBPS available with a maximal theoretical data

rate of 1 Mbps and 2 Mbps respectively. CODED is a long range, low data rate specification[40].

2. **Link Layer:** Interface directly with PHY. Its responsibilities are: Advertising, scanning, and creating or maintaining connection, Encapsulate data from soft device and application layers and generate data packets transmitted by PHY, encryption and decryption, Address management, and define the device role; Advertiser, scanner, master, or slave. When a data packet is sent data is sent to the Link Layer where it is segmented into data packets. A data packet, as illustrated in Figure 3.12, consist of several byte sections. 1 Preamble byte, 4 Access Address bytes, 2 - 257 bytes of data, defined as Protocol Data Unit (PDU), and 3 CRC (Cyclic Redundancy Check) bytes for error checking. When transmitting data packets the PDU section is divided in 5 sub sections. A 2 byte link layer header, 4 bytes L2CAP header, 3 bytes ATT header, 0 to 244 bytes of ATT data payload, and 4 bytes of Message Integrity Check (MIC). Figure 3.12 also illustrate how advertisement packets are structured. PDU consists of 2 bytes Link Layer header, 6 bytes Media Access Control (MAC) address, and 31 bytes of advertisement data[40].

3. **Host Controller Interface (HCI):** is purely a data transport layer between the host layers and the control layers[40].

4. **Logical Link Control and Adaption Layer Protocol (L2CAP):** Multiplex data received from controller layers to the upper layers. This allow multiple connections to share the same PHY[40].

5. **Security Manager Protocol (SMP):** Provide security services for BLE applications. Such services are: Device authentication, device authorization, device privacy, Data integrity, data confidentiality, and data privacy[40].

6. **Attribute Protocol (ATT):** client server protocol where a set, one or more, attributes are presented by a device. An attribute can be thought of as a unit of data, such as measured temperature data, measured pressure data, etc. The attribute protocol define how to transfer the attribute data. The client request data, and server respond by sending data to its client(s). By enabling notification the server can itself initiate data transmission. An attribute consist of four components: Handle, UUID, permissions, and value[data]. ATT is it self mostly used as transport layer that interact with the generic attribute profile (GATT)[40].

7. **Generic Attribute Profile (GATT):** Unlike ATT where different units of data can be seen as spread out over an area, at the same hierarchical level, GATT is a structuring of attributes, services can contain one or more data value called characteristics, and each data value/characteristic contain a value

and an optional descriptor(additional information), which can be a unit of measurements, etc[40].

8. **Generic Access Profile (GAP):** Primarily define devices dicovery -and connection procedures. In addition a simplification of the SMP layer provide options for pairing, creating bonds, and privacy. GAP can configure link layer to use a device as: Peripheral (slave), central (master), broadcaster (advertiser), observer (scanner)[40].

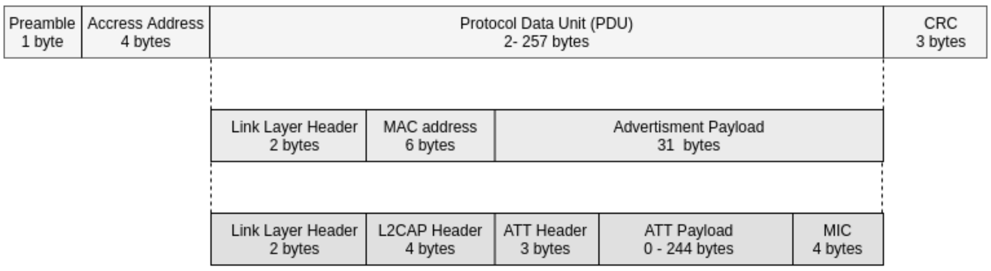


Figure 3.12: BLE data and advertisement packets. Adapted from [40].

3.6 Summary

The system requirements analysis form a basis for the embedded system design and component selection. Figure 3.13 give an overview of the major components and how the are interfaced. All the components are centered around the BL652 MCU -and radio module. Digital sensors are interfaced with I^2C, SD card with SPI, LED with PWM and a power LED driver, and stepper motor is interfaced with UART through an RS3232 controller module. The analog pressure and battery voltage is measured by with the SAADC peripheral in the MCU, where voltage is scaled to a maximum 3,0 V. The motor power header has a 2700 μF electrolytic capacitor, C1, as recommended by the manufacturer to smooth voltage disturbances during operation[31]. The BL652 has Bluetooth radio hardware and a tuned antenna network to interface with a desktop computer and portable devices.

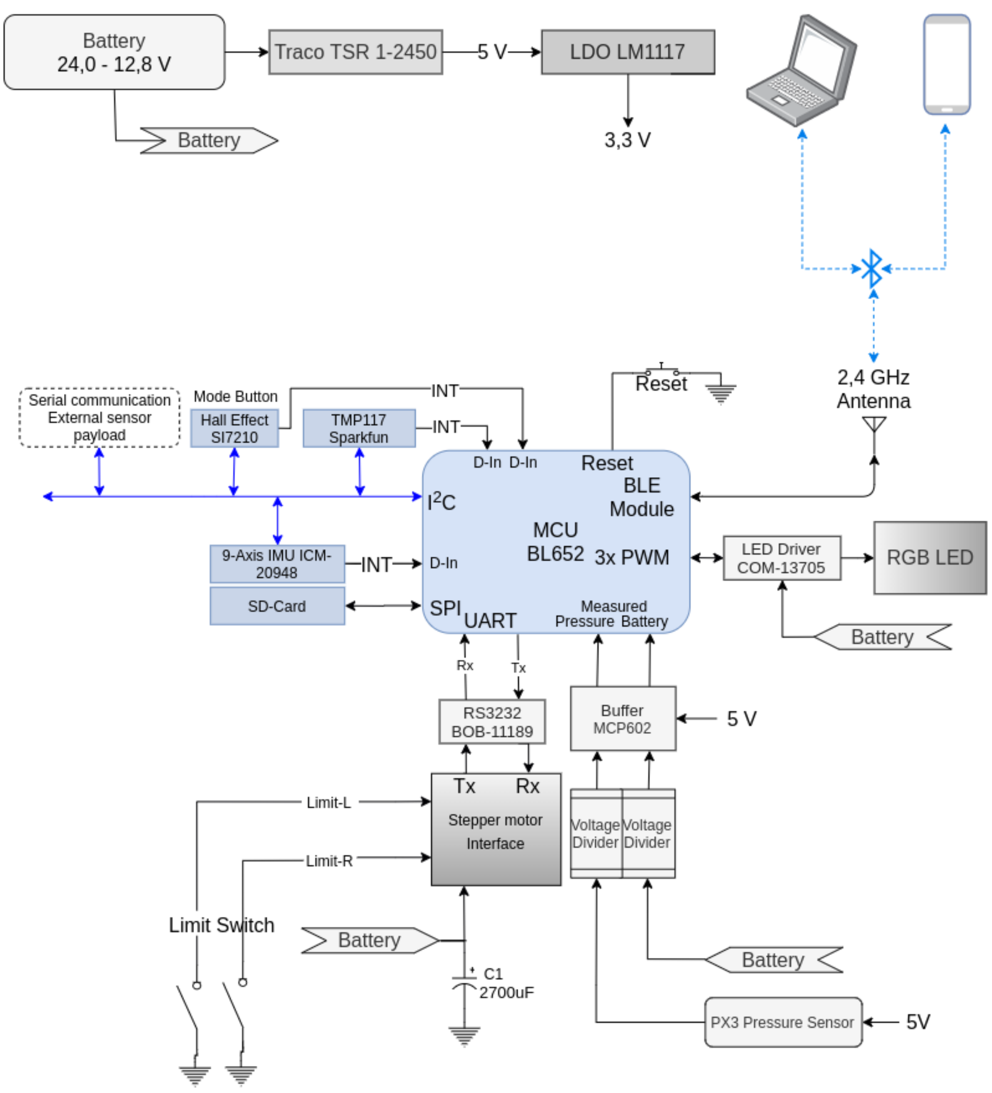


Figure 3.13: Block diagram to give overview of how each major component is connected and interface.

Chapter 4

Printed Circuit Board

This chapter presents and describe tools, design, and fabrication of the printed circuit board (PCB).

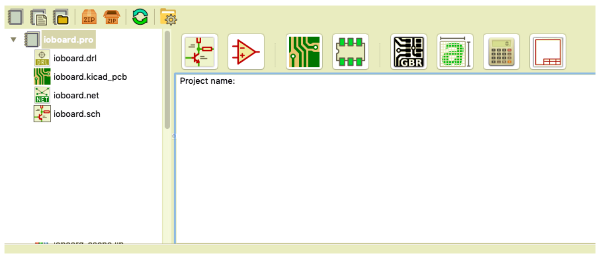
4.1 KiCAD

KiCAD is an open source Computer Aided Design software to design electrical circuit schematics and PCB. Figure 4.1a, b, and c show the main project window, the schematic editor, and PCB layout editor respectively. Its main strength is its versatility in form of cross platform compatibility (available for Windows, Linux, and OS X), its users provide schematic symbol and footprint for a high number of components and modules, and vast community for troubleshooting and user tips. KiCAD also offer to install user designed plugins, such as creating Bill of Materials (BOM), and footprint placement file, useful when placing the PCB order at the manufacturer.

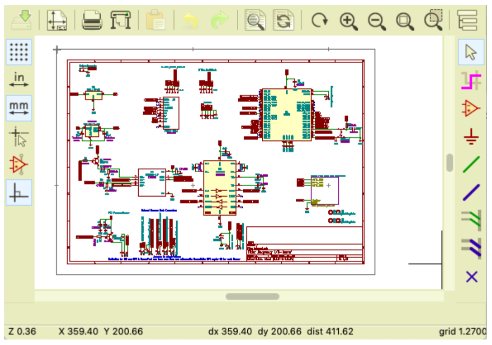
Symbol and footprint: For each schematic symbol, representing a component or module, a footprint is assigned. This will show up as solder pads with corresponding numbered pins to fit the schematic connections.

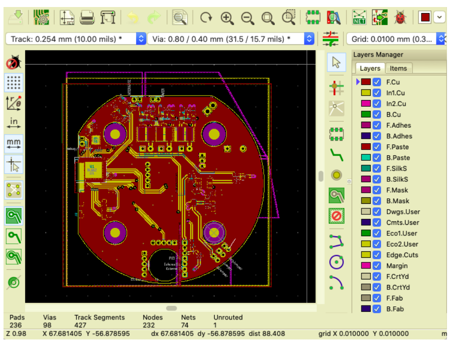
Sheet hierarchy: The schematic layout can be divided in a hierarchical pattern with main sheet and sub-sheets. Sub-sheets have input and output ports that connect to the main sheet. This make a complex schematic easier to design and navigate.

Labels: Labels are used to interconnect components without the need to pull wires between them. This provide a clean and tidy schematic layout. Global labels are accessible in each sheet and sub-sheet, while local labels are only accessible on a given sub-sheet.



(a)





(b) (c)

Figure 4.1: KiCAD - **a:** Project editor **b:** Schematic editor **c:** PCB layout editor.

4.2 Schematic

Standard procedure when designing a PCB is to first create a schematic. It serves as an overview of each component and how components are connected. The schematic is also a prerequisite for the PCB layout. Figure 3.13 above provide an overview of the main components.

This section describe the schematic of each module and its corresponding components. The full schematic is included in Appendix C.

4.2.1 Power Supply

The power supply scheme is displayed in Figure 4.2. The 24 V battery pack is connect to the circuit through a two pin connector header. Positive terminal is connected to a global tag, labeled "Battery", and the other terminal is connected to the global label "GND". The next stage is the Traco TSR_1-2450 switching regulator. It takes the "Battery" as input and has 5 V on the output. Lastly the LM1117-3.3 take 5 V as input and give a stable 3.3 V on the output. In order to provide a smooth 3.3 V output a 10 μF tantalum decoupling capacitor is placed on both input and output side of the regulator. Tantalum capacitors are both smaller and more accurate than electrolytic capacitors.

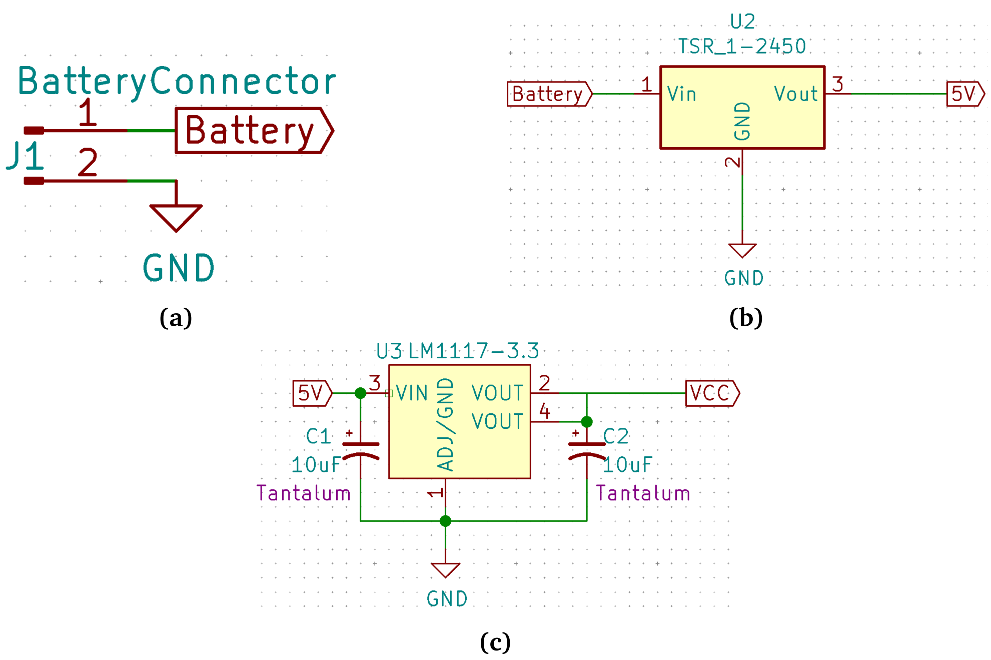


Figure 4.2: Power Supply - **a:** Battery connector **b:** Traco TSR-1-2450 Switching regulator **c:** LM1117 3.3V linear regulator.

4.2.2 BL652 Radio Module

Figure 4.3 show the BL652 radio module schematic symbol with its input and output labels. A two pin connector is attached to SWDCLK and SWDIO to allow for programming and debug after after the module is soldered on the PCB. A push button is connected between Reset-pin and GND such that when pushed the reset is pulled to GND and resets the microcontroller. The reset-pin is has a 10 KΩ pull-up resistor and a 100 nF de-bounds capacitor acting as a shunt capacitor that smoothes out the boundsing effect when the button is pressed.

4.2.3 PX3 Pressure Sensor

The pressure sensor is itself an external component not mounted on the PCB. It is, however, supplied by 5 V from the Traco switching regulator, and its output is read by the BL652 module. Figure 4.4 show the 3 pin connector header, where pin-1 is the sensor output labeled "Pressure", pin-2 is connected to GND, and pin-3 is connected to 5 V through a transistor. The transistor is placed in order to shutdown the sensor when not needed, example when the vehicle is on on-shore and pressure data is not needed. This to hinder unnecessary current from draining the batteries. This transistor is placed in the design as a last-minute modification, without a proper review, and turned out to have a design flaw – explained in subsection 8.1.3. As the sensor may be affected by the regulator switching noise two ceramic decoupling capacitors are placed in front of the transistor collector,

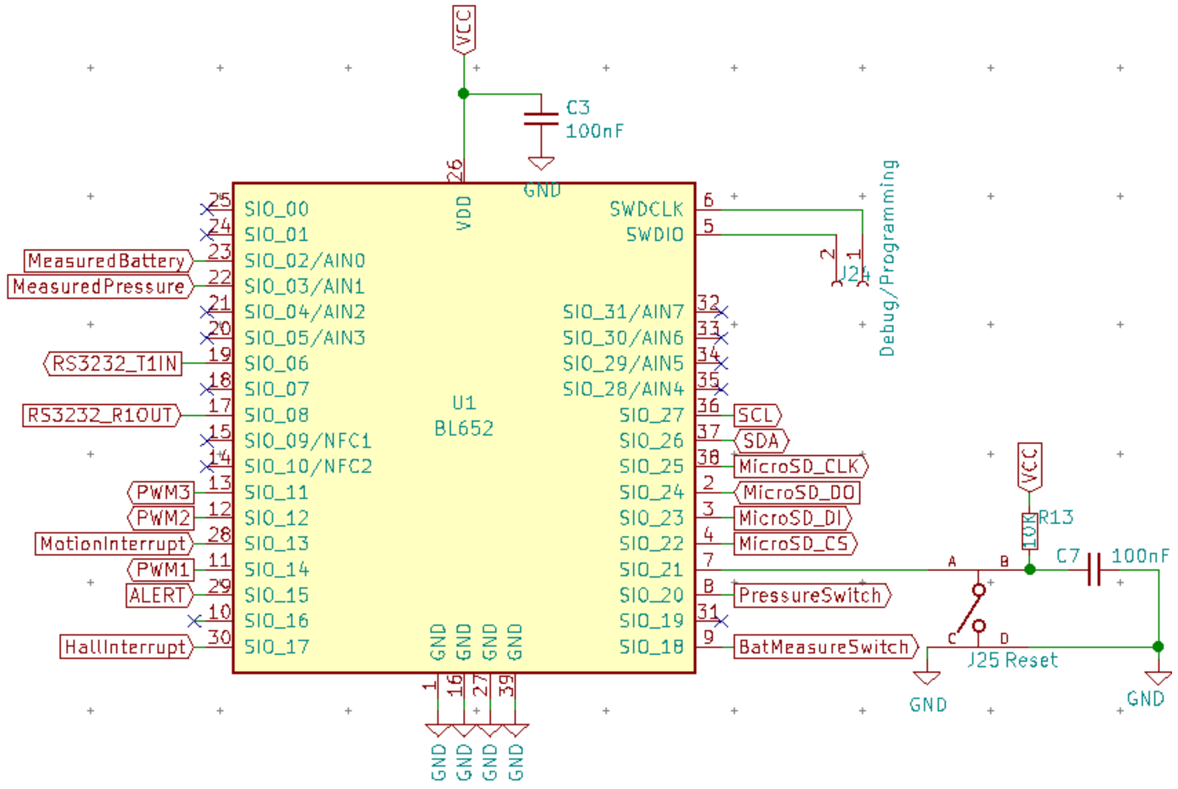


Figure 4.3: BL652 Radio Module with input and output signal labels, SWD debug headers, and reset switch.

to shunt some of the high switching frequencies to GND.

4.2.4 MCP602 Unity-Gain Buffer

The MCP602, Figure 4.5, has two package has two operational amplifiers in it, A and B. Input VinA+ has a voltage divider scaled battery voltage, that fits within the nRF52832 SAADC range. It also has a transistor to cut off the battery drainage through the voltage divider when battery measurement is not necessary. This transistor is placed in the design as a last-minute modification, without a proper review, and turned out to have a design flaw – explained in subsection 8.1.3. The voltage is supplied on the output VOUTA and labeled "MeasuredBattery". Input VinA- has the label as input. Input VinB+ has the PX3 pressure data as input, scaled slightly by a voltage divider to fit within the SAADC range. As for opamp A the output of op-amp B is fed back to VinB- to configure it as unity-gain buffer. The output of op-amp A and B are both measured by the radio module ADC.

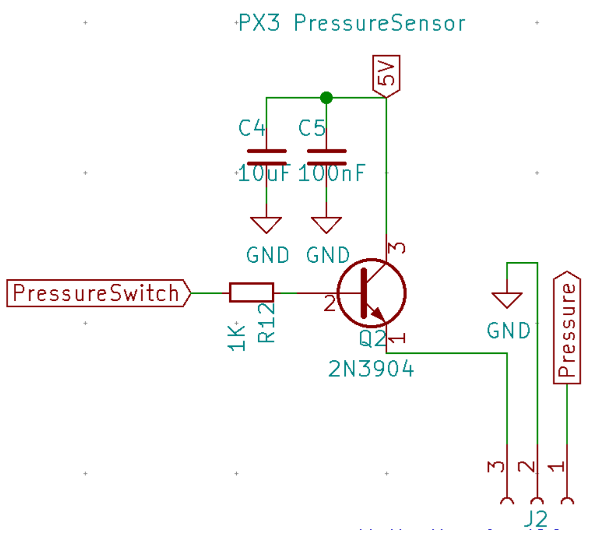


Figure 4.4: PX3 pressure sensor headers, with C4 and C5 shunt capacitors to filter out high frequencies, and trans to switch on/off power.

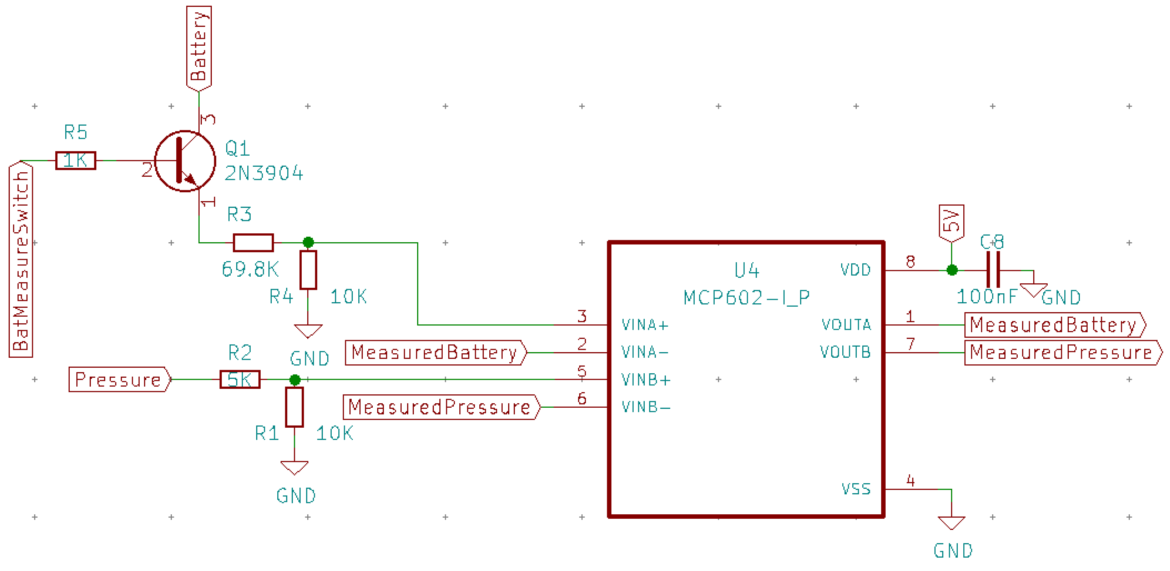


Figure 4.5: MCP602 configured as a unity-gain buffer for the measured battery and pressure voltage.

4.2.5 Motor Power and Interface

The motor is connected to the battery voltage through a 2 pin header, as can be seen in Figure 4.6. Between the positive terminal and GND is a 2700 μF capacitor to hinder instabilities when the motor is in operation.

The motor is interfaced to the microcontroller through RS232. The RS3232 module, Figure 4.7, is powered by 3.3V. and is interfaced to the microcontroller over UART.

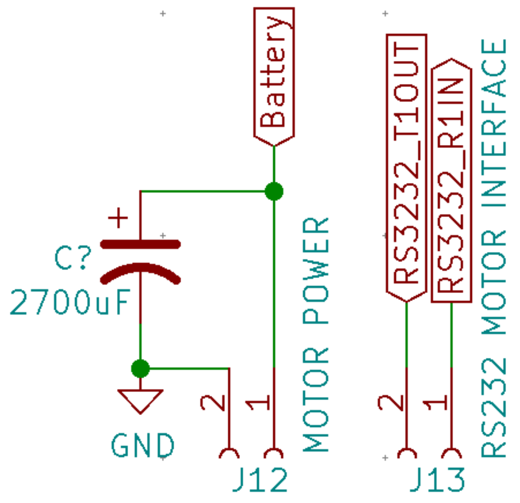


Figure 4.6: Motor power and RS3232 Rx/Tx headers.

4.2.6 SD Card Socket, and Sensor Headers

Figure 4.8 show the pin headers for the 9-axis motion sensor mounted on the PIM448 module, the Si7210-Hall effect module, TMP117 temperature module, and the SDcard socket. The motion sensor module is placed on the PCB. The Si7210 and TMP117 modules are placed at the lower half of the vehicle housing, and need header pins to connect to the PCB. The SD card socket is a surface mount component that is soldered on the PCB. Each sensor have one interrupt pin, and is interfaced with I^{2}C. The SDcard socket is interfaced with SPI.

Figure 4.9 show the headers dedicated for a future external sensor package. The communication protocol chosen is I^{2}C as no additional wiring is needed when more sensors are added, in contrast to SPI where each sensor need an extra chip-select pin. It is also not know what voltage level these sensors need. The three voltage levels are therefor supplied with a jumper system so that the desired voltage level can be supplied to the sensor pack voltage pin, labeled "External-SensorPackVoltage".

4.2.7 PicoBuck v12 LED Driver

As the picobuck LED driver is a relatively large module it is decided to integrate the module as individual components to the PCB. This is to have greater flexibility for placing the components on the PCB. The schematic is created by Sparkfun in collaboration with Ethan Zonca, and downloaded from the following github repository: https://github.com/sparkfun/PicoBuck. It is licensed under Creative Commons Share-alike 3.0: https://creativecommons.org/licenses/by-sa/3.0/

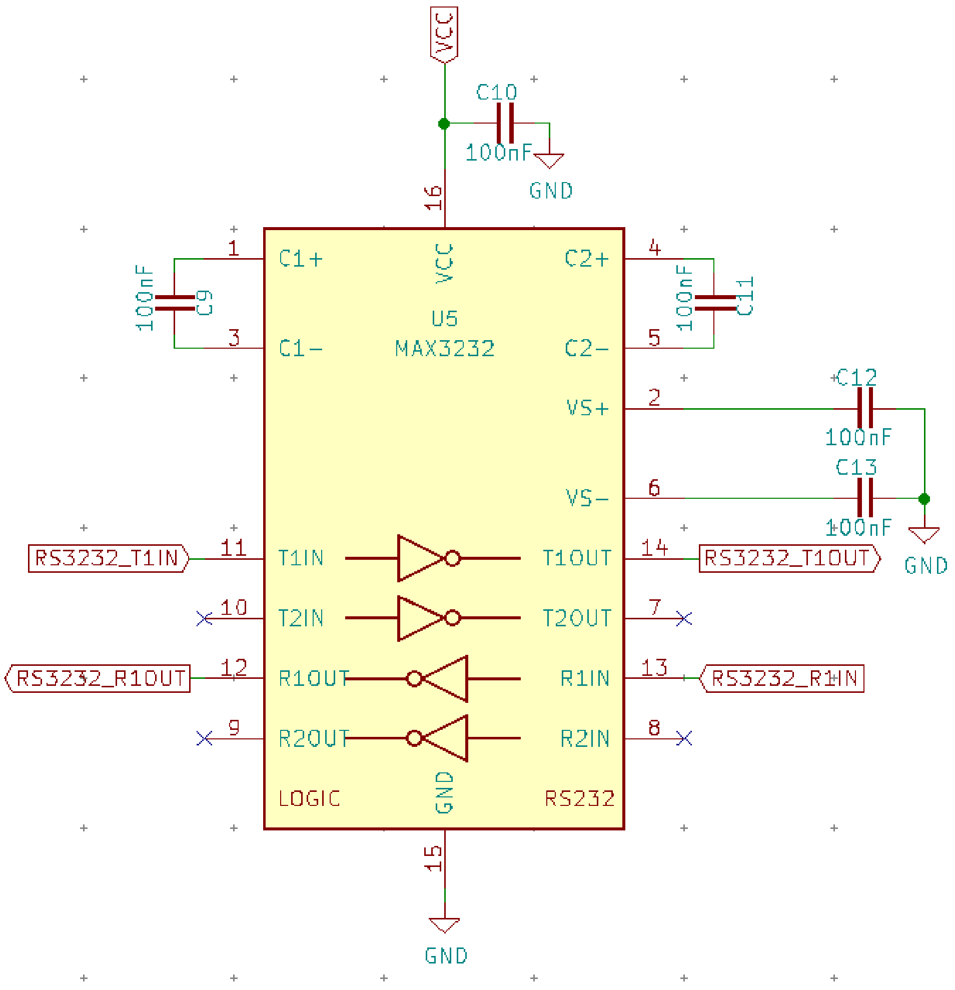


Figure 4.7: RS3232 chip with booster capacitors and interfacing between BL652 and Motor controller module.

The Picobuck schematic is placed in a sub-sheet as it is a standalone module with quite a few components that repeat for each channel. Figure 4.10 show the symbol of the sub-sheet in the main sheet. It takes PWM1, PWM2, PWM3, and Battery as input, and GND as output. Although, since Battery and GND are global labels it is not strictly necessary to give them as input, but it makes it easier to follow the signal flow. Figure 4.11 show the schematic of a single picobuck channel, and Figure 4.12 all three channels.

A couple of changes are made to fit my need. One, three headers are removed, as the PWM, GND, and battery connections are now directly connected to components on the PCB. And since as JLCPCB manufacturer does not have all the compon-

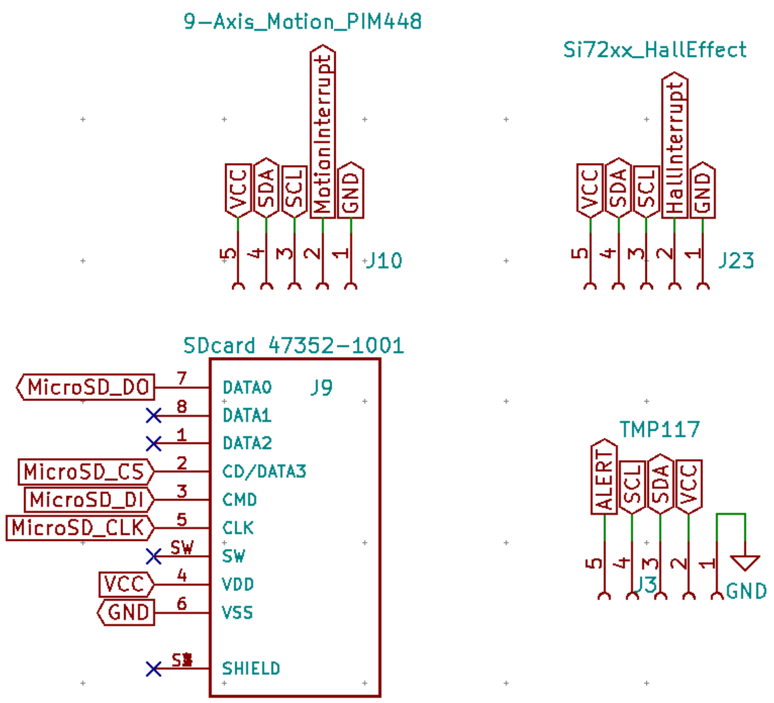


Figure 4.8: SD card socket and sensor headers for TMP117, PIM448 motion module, and Hall effect sensor

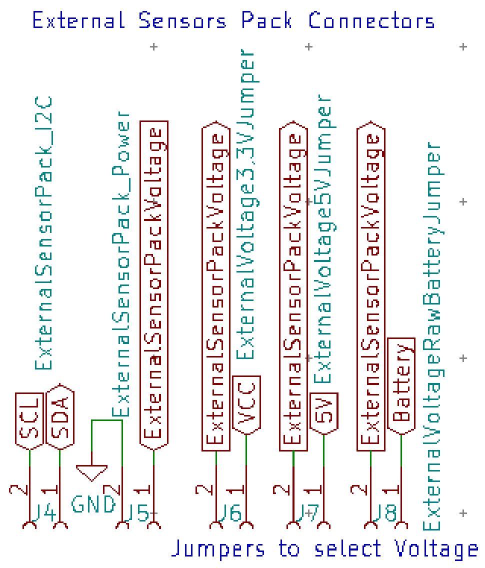

External Sensors Pack Connectors
ExternalSensorPack_I2C
SCL
SDA
J4
GND
ExternalSensorPack_Power
ExternalSensorPackVoltage
J5
ExternalSensorPackVoltage
ExternalVoltage3,3VJumper
VCC
J6
ExternalSensorPackVoltage
ExternalVoltage5VJumper
5V
J7
ExternalSensorPackVoltage
Battery
J8
ExternalVoltageRawBatteryJumper
Jumpers to select Voltage

Figure 4.9: External sensor headers, and jumper headers to select voltage level: 3,3 V, 5 V, or raw battery.

ents available, a few components are changed to fit JLCPCB available components.

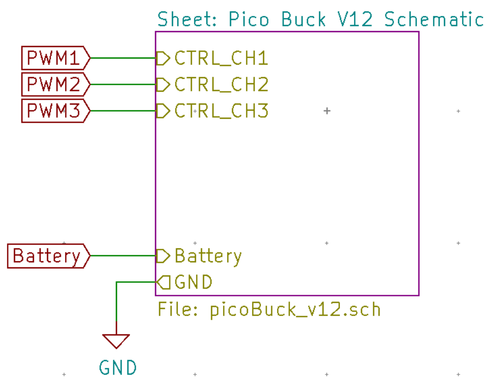


Figure 4.10: PicoBuck v12 sub-sheet symbol.

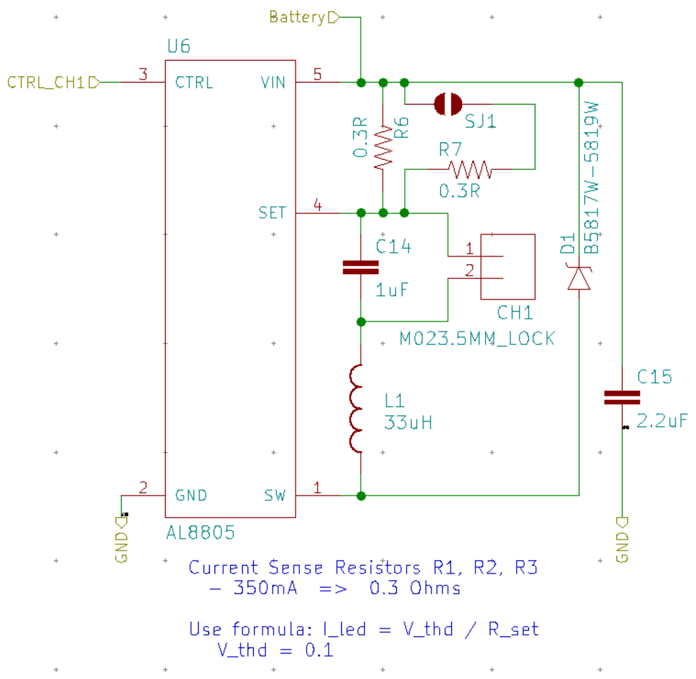


Figure 4.11: One channel of the PicoBuck module.

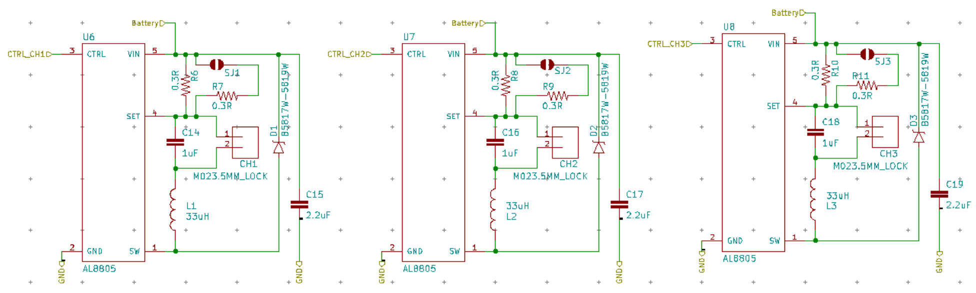


Figure 4.12: All three channels of the PicoBuck module.

4.2.8 Mounting Holes

The PCB is mounted with four M4 bolts. The dimension on the mounting holes are set to be 4.3 mm in diameter.

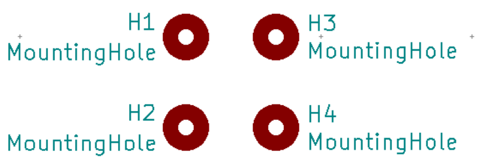


Figure 4.13: Schematic representation of 4 PCB mounting holes.

4.3 Layout

4.3.1 Group Components

First thing when laying out a PCB design is to import all the component footprints from schematic design. Then group components together into voltage levels and again group components into main components and surround them with support components, such as resistors and capacitors. This is to start off with a clear picture of how what components are critical to place in proximity to each other, and which ones can be separated. Grouping components based on voltage levels also help planning and dividing power plane into approximately the right size from the start.

4.3.2 Replace Modules With Components

In an attempt to save time and money, the Initially plan was to design the PCB mainly as an I/O board where the different modules were mounted on the PCB, thereby alleviating the need for large amounts of surface mount components, and the surface mount process it self. However, it turned out difficult to realize and

I/O board consisting only of modules as the circular shape and the four monting holes took up too much board area to fit all the modules. It was therefor decided to replace to ready made LED driver module, the RS3232 module, and SDcard mount as individual surface mount components. While the ICM motion sensor module and BL652 radio module remained mounted as modules, mainly because of inconvenience in finding footprint and the individual parts.

4.3.3 PCB Edge Cuts

The PCB is shaped by lines, curves, and circles in the Edge.Cuts kiCad layer in order to provide information to the manufacturer how the PCB is cut and shaped. The PCB must fit inside the circular inner lid with diameter 98 mm. The PCB diameter is set to 95 mm to leave some extra room. The PCB also need to leave room for the radio module antenna to stick out from the PCB edge, another side have wires for battery, motor power, and RS232 interface. A 15 mm circular hole is made along the PCB edge to fit an equally sized hole through to inner lid to give space for wires from Temperature sensor, Hall Effect Sensor, and pressure sensor. Figure 4.14 show the finished Edge.Cuts layer. The various connectors are placed along these edges for easy access.

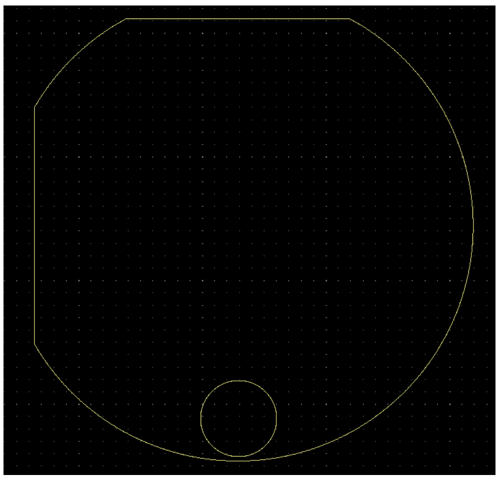


Figure 4.14: KiCAD Edge.Cuts layer shows the PCB shape

4.3.4 Mounting Holes

The mounting holes are placed according to drawings of the inner lid to fit the M4 motor fastening screws. Screw holes have 0.3 mm clearance to give some room for the screws to fit without the need to force the PCB in place. a 10 mm pad clearance is made for the solder mask, and an additional 1 mm clearance for the copper layer, in order to give room for a 10 mm washer without risking to scrape off the solder mask short the inner layers through the metal screw. See Figure 4.15 and Figure 4.16.

4.3.5 4 Layer Design

The four layer PCB design, shown in Figure 4.15, is divided such that the top and bottom layers are dedicated to signal traces, and is covered with ground plane. The ground plane is there for three specific reasons, according to [42]:

1. To provide shielding from noise.
2. To provide sufficient grounding.
3. To provide sufficient heat dissipation on both sides of the PCB - in order to avoid the PCB from bending under thermal stress, as the material changes temperature unevenly on each side.

Layer 2 is a dedicated GND plane. Layer 3 is a dedicated power plane.

4.3.6 Power Plane

The PCB has three voltage levels; battery voltage (24 V - 12.8 V), 5 V, and 3.3 V. Figure 4.15c show how the power plane is divided into three power domains, where each voltage level supports components with the needed voltage level. To avoid noise, components of each voltage level is placed above its own power domain, in order that wiring a voltage supply through a different voltage level is avoided as best as possible.

4.3.7 Ground Plane

Similarly as the divided power plane ground planes are also divided. though not entirely separated, into domains, see Figure 4.15a,b, and d. This is to avoid, as best possible, the switching regulator and motor noise from interfering with the weaker 3.3 V circuitry. Barriers of "keep-out zones" are laid out on all three ground layer planes to create an edge for which noise will bounds off of, and hopefully find its way to the negative battery pole, instead of reaching and creating disturbance throughout the 3.3 V ground area.

4.3.8 Wiring and Via Holes

Two widths are used for Wiring. Signal wires are grid 10 mil, VCC and GND are 25 mil. Exceptions are made for VCC and ground on the radio module and RS3232, as pads are too small for 25 mil traces. The VCC and GND pins of connectors are powered directly from planes as these are through hole connectors that connect to power, or GND as needed.
A number of via holes are placed all over the board in order to lead signal between top and bottom layers, provide power and GND to surface mount components, and to ensure good ground connection to all the parts of the board.

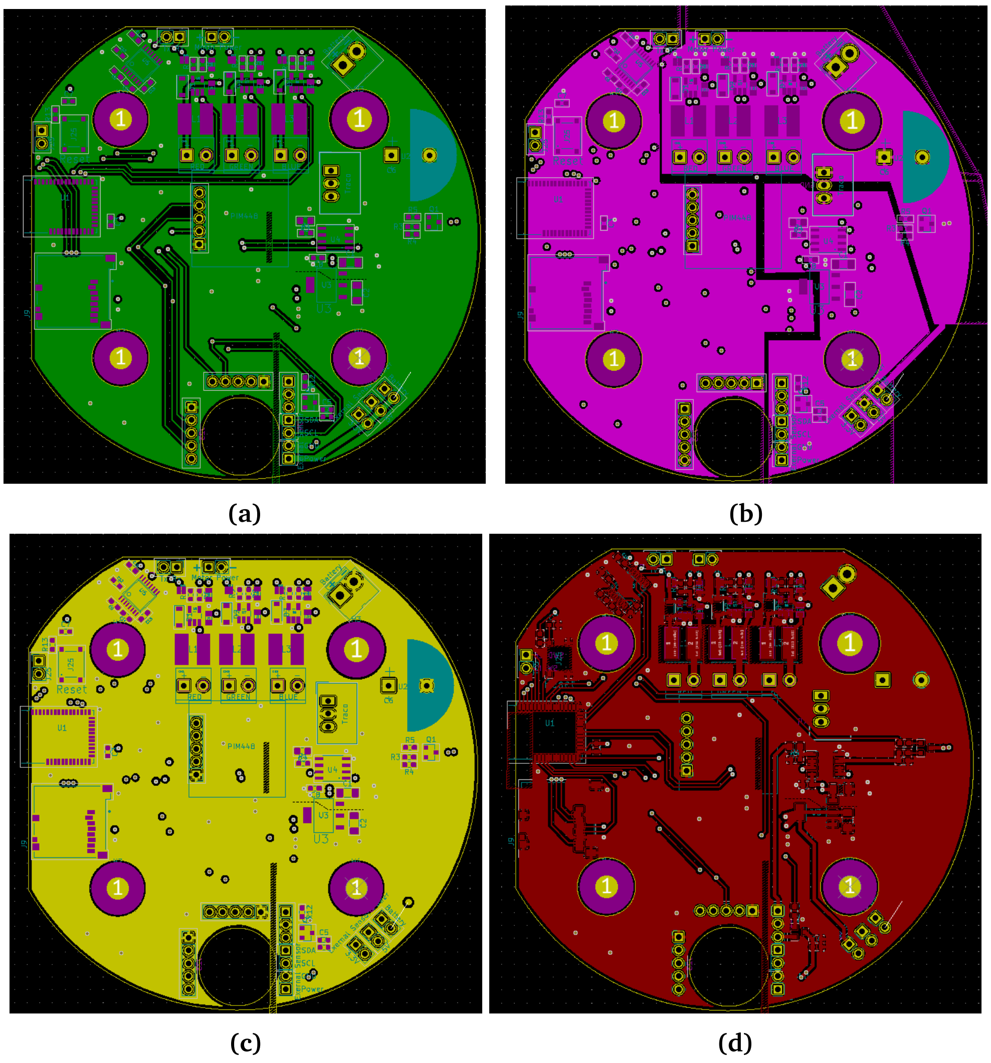
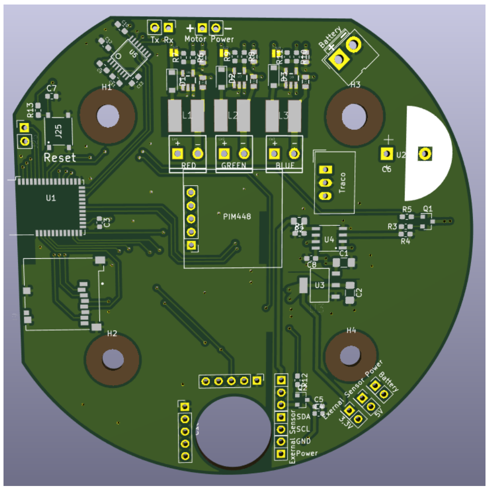


Figure 4.15: 4 layer PCB - **a:** Top signal plane **b:** Ground plane **c:** Power plane **d:** Bottom signal plane

4.3.9 Silkscreen

Figure 4.16a and b show the finished PCB design. Text is written to provide information of where each component and module is to be placed, and what set of pins belong to a given sensor. As well as the function of a specific header pin. The text is written on the top and bottom silkscreen layer.


(a)

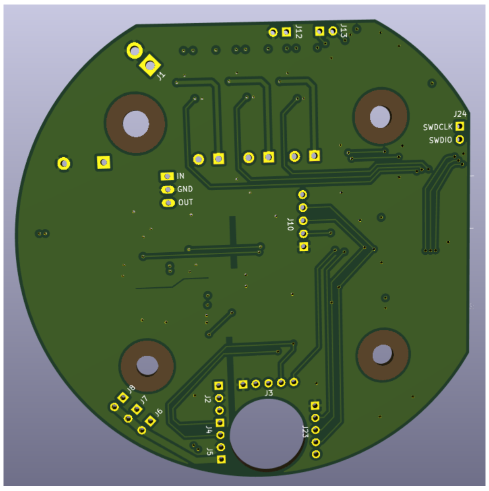

(b)

Figure 4.16: Finished PCB design **a:** Top side **b:** Bottom side

4.4 Manufacturing

The PCB is manufactured at JLCPCB, based in China. JLCPCB offer solder surface mounting of components available in their database of 30 000 components. As most components can be found in their database this provide a time efficient solution for placing and soldering components.
when ordering the PCB with components a PCB plot file is generated by kiCad, and a component placement file is generated in kiCad by installing a plugin specifically for ordering surface mount PCB designs from JLCPCB. THe plugin is called JLCkicadtools, created by Matthew Lai, and available at `https://github.com/matthewlai/JLCKicadTools`.

4.4.1 Adjust Misaligned Components

The CPL file generated by the program is mostly successful at rotating and aligning components correctly. However, there are cases where components need to be aligned manually by accessing the file and adjusting the rotation variable 'Rot'. Figure 4.17 show how JLCPCB ordering page has misaligned parts. The misaligned parts are identified by their legs not being aligned with the footprint. Figure 4.18 shows a section of the CPL file, where the Rot value is adjusted to align components to PCB footprint. If JLCPCB show a misaligned component, example Q1 and Q2 in Figure 4.17, then Rot value of the specific components need to be adjusted. Q1 and Q2 need to be adjusted by 180°.

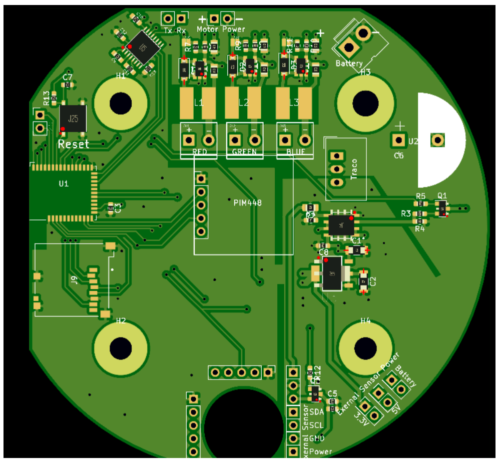
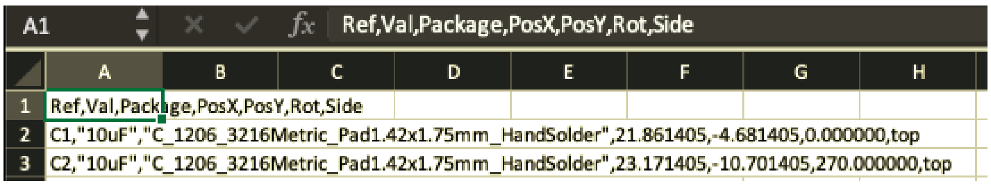


Figure 4.17: When placing order at JLC surface mount assembly services the rotation may be incorrect, and need to be rotated manually by editing the rotation variable 'Rot' in CPL file.



A1 fx Ref,Val,Package,PosX,PosY,Rot,Side

	A	B	C	D	E	F	G	H
1	Ref,Val,Package,PosX,PosY,Rot,Side							
2	C1,"10uF","C_1206_3216Metric_Pad1.42x1.75mm_HandSolder",21.861405,-4.681405,0.000000,top							
3	C2,"10uF","C_1206_3216Metric_Pad1.42x1.75mm_HandSolder",23.171405,-10.701405,270.000000,top							

Figure 4.18: A section of the CPL file where the rotation variable 'Rot' is adjusted to align the components correctly to PCB footprint

4.5 PCB Revision 2

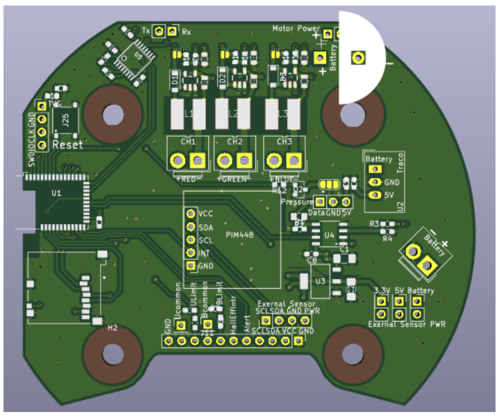
A second revision of the PCB is made after the first design is received and thoroughly tested with the rest of the system. The most significant changes are related to the layout of the header connectors and the edge cut (physical shape).

The finished layout is shown in Figure 4.19. The white front silk of the capacitor shown in upper right corner is not removed as the library had to be edited. It has no consequence for the manufactured PCB. The edge cut is modified from a 15 mm hole to an open area, to fit a section cut out of the inner lid described in section 6.5, to better handle wires. the BL652 has a keep-out area underneath and around the chip antenna. The PCB edge-cut is shaped to fit the keep-out area of the module in order to better the wireless signal quality. The headers are aligned to fit close together in groups. The wires coming up from the sensors and end limit switches are placed in a row at the bottom of the layout, as seen from Figure 4.19.

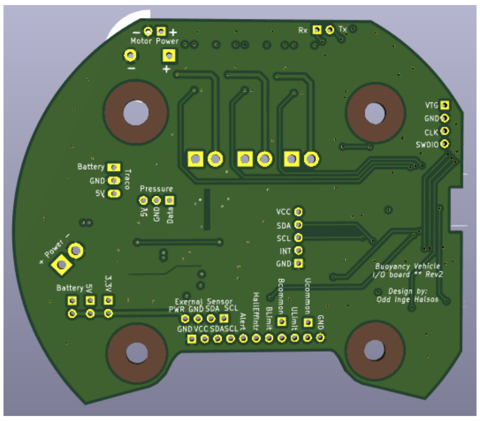
The battery connector and the 2700 μF capacitor have switched places in order to fit the motor power headers and the capacitor closer together. The LED headers are changed from 2,54 mm header pins to 3,96 mm as initially intended in order to fit the 3,96 clip-on connectors. This is so that the LED wires don't fall out when outer lid is mounted, adjusted, and unmounted. To allow for the larger connectors to fit in the PIM448 module headers are moved 5 mm down.

The pressure sensor headers are moved to fit between the traco 5 V regulator and the MCP602 op-amp. The mounting hole pad and solder mask clearance are expanded by 0.5 mm in order for the washers to fit within the clearance. This in turn made it necessary to re-arrange the capacitors around the RS3232 module a bit.

The schematic of BL652 module is shown in Figure 4.20. A GND and VDD header was added together with the SWD IO and SWD CLK headers to make connection for programming easier to set up. The SW CardDetect pin on the SD card module is connected to the BL652 module in case of a future interest to detect whether an SD card is mounted or not. The transistor switching on and of power to the pressure sensor is moved from the VDD header to the GND header, in order to switch on and of the connection to GND and thereby control when the sensor is operating, in the hopes of saving a few milliamps of static power consumption. A solder pad jumper is added so that if the transistor is no longer wanted in the system one can simply solder the middle and right jumper pads to connect GND and sensor GND directly, bypassing the transistor. The middle and left pad is connected by trace even though they are not visibly soldered. The transistor intended to disconnect battery power from the measured battery voltage divider is removed from the design. Finally as the voltage on the limit switch pins are 4,2 V when limit switch is closed, a voltage divider is added for the bottom closed switch and upper closed switch pins. This regulate the voltage to below 3 V so that the pin can be connect to input pins on the BL652 module. These are used to generate interrupt events whenever bottom or upper limit switch are reached. The full schematic of revision 2 is included in Appendix D.



(a) PCB second revision top side


Motor Power
Battery
Pressure
Power
Buoyancy Vehicle I/O board ** Rev2
Design by: Odd Inge Halsos

(b) PCB second revision bottom side

Figure 4.19: Finished PCB revision 2 design - **a:** Top side **b:** Bottom side

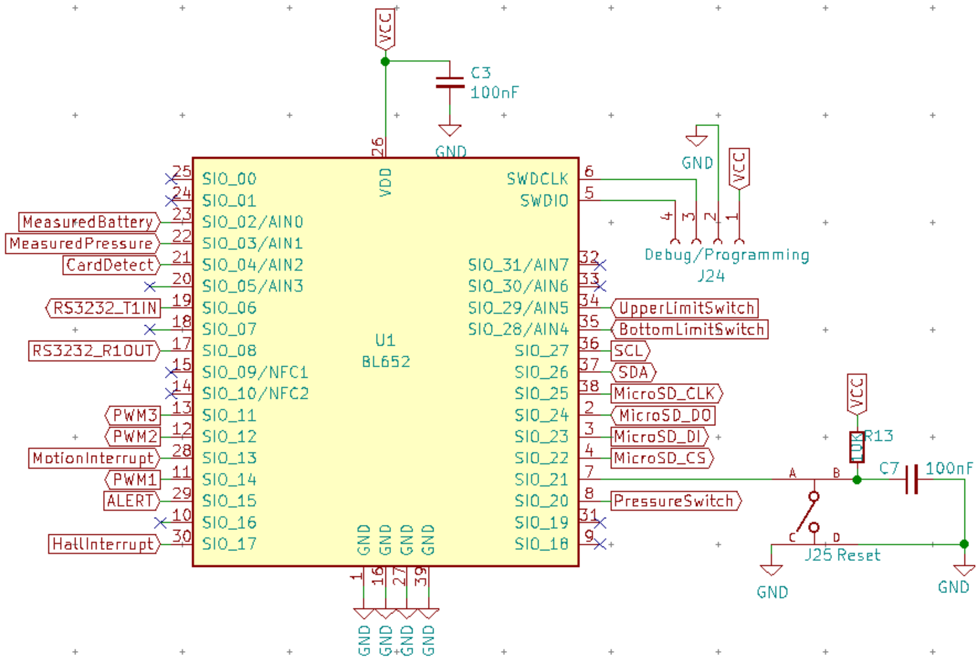


Figure 4.20: Schematic revision 2: BL652 Radio module with input and output signal labels, SWD debug headers, and reset switch.

Chapter 5

Firmware

Firmware is a vital part of any embedded system. Firmware can be thought of as the mental activity of the microcontroller (the brain), that interface the other hardware components. Firmware is an essential, and by far the most time consuming, part of the project. Much time is devoted to create appropriate firmware in order to fulfill necessary use cases. This chapter describe important aspects of the code-base to provide an overview and understanding for how the system works, and how the components are tied together. Additional theory is provided in section 5.10 to calculate velocity.

5.1 Embedded Software

The embedded software stack is illustrated in Figure 5.1. The Hardware layer is at the bottom of the stack, which is the physical hardware (peripherals, radio, etc) of the nRF52832 SoC. The application code at the upper level can access the hardware peripherals by creating 'bare bone' peripheral drivers, or including existing drivers made from either the Nordic team, or external contributors. In order to access the radio hardware and create Bluetooth® services a SoftDevice need to be flashed onto the chip.

5.1.1 BLE Protocol Stack: SoftDevice

SoftDevice is Nordics proprietary wireless protocol stack. It handles everything happening underneath the application level related to wireless communication. The SoftDevice notify the application layer of events from subsequent layers when taken place, so that application can handle it.[43] The SoftDevice developed for nRF52832 is S132, with latest version being v7.0.1. It come as a pre-compiled and linked binary file, which can be downloaded from:[38]

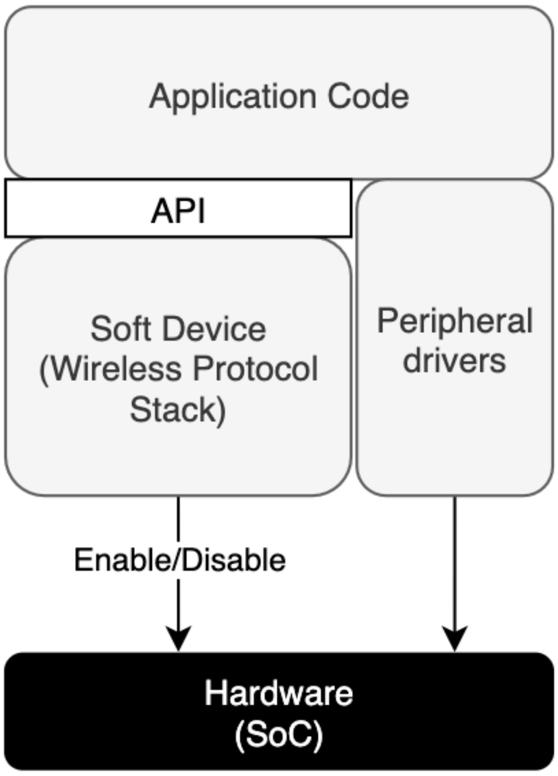


Figure 5.1: Nordic Semiconductor embedded software stack. Developed application code interface with the proprietary SoftDevice module through provided API. Hardware layer is accessed by SoftDevice when enabled, and application code through peripheral drivers

5.1.2 Software Development Kit

The Software Develeopment Kit (SDK), provided by Nordic Semiconductor, is a collection of drivers, modules, and documentation for each specific chipset family. This project, based on the nRF52832 SoC, need the nRF5 SDK v16.0.0, which can be downloaded from: `https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK/Download`.

each driver and module included in the nRF5 SDK is implemented following a well defined set of operations, where each operation is described in library/module section of the SDK documentation.

The typical implementation of drivers and modules are outlined as follows:

1. Create instance of the driver/module, with an instance name and additional parameters
2. create an interrupt handler function, which is called when the SoftDevice generate event interrupt related to the driver/module
3. populate the function with appropriate handling of the events releated to

the driver/module. A list of relevant events are found in the SDK documentation.

4. create a configuration structure instance and populate it with relevant data
5. call the initialization function with instance name, configuration instance, and handler function
6. get familiar with relevant functions associated with the driver/module and create new functions calling these when appropriate.

Note: Functions related to a specific instance must be called inside the same file as the instantiation, as each function need to pass the address of the instance name to the function. To use driver/module functions outside of the scope of a given file, an additional function must be created which again is calling the driver/module function. Declare this/these functions in the header file and include the header file in the .c file where you want to call the function from. The benefit here is that each module has a clearly defined scope, while still open to be accessed by specific functions intended for this purpose.

5.2 Bluetooth Low Energy

This section will give a brief overview of important topics related to the implemented Bluetooth Low Energy firmware.

5.2.1 Bluetooth Configuration

The application is based off of an SDK BLE peripheral example. All the necessary BLE configurations are set to a working condition. However better performance or lower power consumption can be achieved by making changes to certain configuration parameters.

The first two parameters are found in the config.h, that hold important configuration data definitions used in the project. The `NRF_SDH_BLE_GAP_DATA_LENGTH`, the maximal GAP length of a transmitted package, is initially set to 251, which is maximal achievable GAP data length. As illustrated in Figure 3.12, the PDU can hold 257 bytes, where 6 bytes are occupied by Link Layer and L2CAP Headers, which leave 251 bytes for the GAP data, with 3 ATT Headers, 4 bytes of MIC 244 bytes of pure data payload. `NRF_SDH_BLE_GATT_MAX_MTU_SIZE` set the maximal internal transportation unit to 247. This is so that the whole data packet, which does not include the 4 MIC bytes are transported at once. during client/server connection the data length and Maximum Transmission Unit (MTU) is negotiated to find the best data exchange rate. It is observed that when connecting to a mac OS X client the negotiated data length is 101 bytes, and MTU is negotiated to 104 bytes. Data length refer to GAP length without the 3 byte ATT Header or 4 bytes of MIC. As the negotiation result in less than maximal MTU and data length the client side is the limiting factor in data throughput. However, the connection interval, which define the time period between each initiated connection, where one or more (up to 6) packages are sent, is initially set to 20 ms. PHY is a major contributor to power consumption and connection interval should be held as high as possible to save power. In this case, however, it is chosen to sacrifice power to achieve better data transmission rate by lowering the connection interval from 20 ms to the minimum possible interval at 7,5 ms.

During configuration it is found a bug when transferring file from vehicle to the BuoyancyApp where only parts of the message is received. This behavior is unique for BuoyancyApp and is not a case when transferring the same file with the same settings on a mobile device. Changing `NRF_SDH_BLE_GAP_DATA_LENGTH` to 101, and `NRF_SDH_BLE_GATT_MAX_MTU_SIZE` to 104 alleviate the problem, but introduce a bottleneck in transmission rate as other device may want to negotiate higher GAP data length and MTU size values. Other key configuration values:

- `#define DEVICE_NAME "Buoyancy"`: The visible advertisement name seen by a scanner device.
- `#define APP_ADV_INTERVAL 64`: The advertisement interval in units of 0,625 ms, which equate to 40 ms between each advertisement package is sent.
- `#define APP_ADV_DURATION 18000`: The advertisement duration in units of 10 ms, which equate to 180 seconds.

5.2.2 Custom Service: Nordic UART Service

Communication between the vehicle and control device (PC, mobile phone, or tablet) is based on the Nordic UART Service (NUS). NUS is a proprietary custom service (not restricted to SIG standard service) with a 128-bit vendor specific Universally Unique Identifier (UUID), generated for each specific application. This allow the developer to tailor the service to fit the applications specific need. The NUS UUID is already given by the SDK example, and it is chosen to leave it unchanged. In the case where the prototype evolve to a production type product the UUID have to be unique in order to avoid potential conflicting devices broadcasting the same SDK example UUID address.

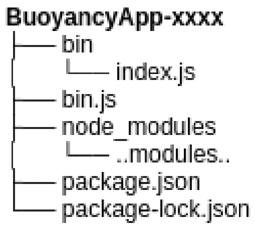
NUS is a custom GATT-based service with receive (RX) and transmit (TX) characteristics to make a simple resemblance of the UART protocol. Data received from the peer, in this case the control device, is passed to the application by calling an interrupt to trigger the handler function, which then collect the data from the GATT write register. When connection is established the 'notify' is enabled which sends data from application through link layer, to the peer. This provide an effective and flexible, yet simple platform for creating a Bluetooth control and data transmission interface.

5.2.3 Desktop Application

A terminal application is developed to interface the buoyancy vehicle. The application is developed in the javascript framework NodeJS. The application is inspired by and built upon the BleTerm2, with permission from the creator Joe Schneider[44].

The application is developed and tested on mac OS X only, as no computer with BLE peripheral running windows or Linux operating system has been available during development and testing. However the application does build and run on windows and Linux, though error message occur due to the lack of proper BLE hardware peripheral. The application use noble-mac Application Programming Interface (API)[45] to access the OSX bluetooth drivers. The windows and Linux version of the application use the noble API[46] to access Bluetooth drivers.

Figure 5.2 show the file tree structure of the application. -xxxx is the -mac or -WinLinux version. The application source code is in the 'Index.js'-file in the 'bin'-folder. 'bin.js' serve as a starter file for the application. node_modules is a folder that contain dependency modules need by the application. 'Package.json' define project properties, description, author, dependencies, etc. 'Package-lock.json' is used to lock dependencies to the current version number, to avoid conflict as application and dependency versions evolve.



```
BuoyancyApp-xxxx
├── bin
│   └── index.js
├── bin.js
├── node_modules
│   └── ..modules..
├── package.json
└── package-lock.json
```

Figure 5.2: Buoyancy application file tree structure

The application is installed on the system by first installing NodeJS from `https://nodejs.org/en/`, then enter the application folder for either mac OSX, Windows, or Linux depending on the platform in use. In the terminal type 'npm install -g' to install the application globally. On successful installation the application is launched from anywhere in terminal by typing 'buoyancyapp'. For connecting to a specific device, type in 'buoyancyapp <device name>', or for a custom UUID, type in 'buoyancyapp <UUID>'. If device identification is typed in the application will scan for the buoyancy vehicle, named 'buoyancy'. To quit application, type 'quit'.

If Bluetooth is not found, or is turned off the following message is printed to terminal: 'Scanning stopped - is Bluetooth adapter connected / turned on?'.

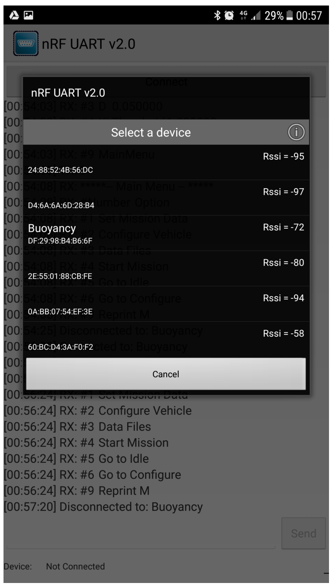
On connection to device a folder is created to hold transfered missionLog files.

The path to the folder is, relative to the application path: '../BuoyancyLog/missionLog' on linux. Or as the application is run from root directory on OSX the folder path is: '/Documents/BuoyancyLog/missionLog'.
The Main Menu is displayed in the terminal window, and navigated with number keys (0-9). A detaile description of the menu is given in section 5.6

The source code is commented according to doxygen standard.

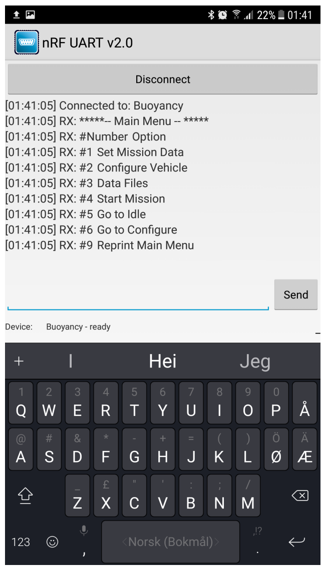
5.2.4 Mobile Application

As no custom application is developed for android or iOS a third party application must be used to interface the buoyancy vehicle with a phone or tablet. Applications such as *nRF UART v2.0*[47] and *Serial Bluetooth Terminal*[48] emulate UART interface over BLE. As the entire text based menu is written from server to client there are no need for a custom application to operate the vehicle.
Figure 5.3 a and b show screenshoot from the nRF UART v2.0 before and after connecting to Buoyancy vehicle. Figure 5.4 a and b show screenshot from Serial Bluetooth Terminal before and after connection is established File transmission is a bit more cumbersome than for the custom made desktop application, as transferred data is not saved to a file. The transferred data is, however, printed to the application terminal window and can be either copied and pasted to a .txt file, or a log can be saved and later edited in a text editor to extract the data of interest.



```
24:88:52:4B:56:DC          Rssi = -95
D4:6A:6A:6D:28:B4          Rssi = -97
Buoyancy
DF:29:98:B4:B6:6F          Rssi = -72
2E:55:01:88:CB:FE          Rssi = -80
0A:BB:07:54:EF:3E          Rssi = -94
60:BC:D4:3A:F0:F2          Rssi = -58

[00:56:24] RX: #2 Configure Vehicle
[00:56:24] RX: #3 Data Files
[00:56:24] RX: #4 Start Mission
[00:56:24] RX: #5 Go to Idle
[00:56:24] RX: #6 Go to Configure
[00:56:24] RX: #9 Reprint M
[00:57:20] Disconnected to: Buoyancy

Device:   Not Connected
```

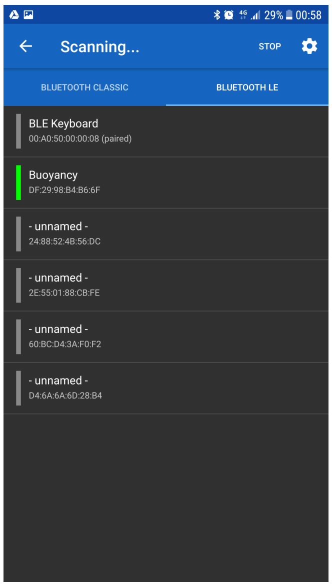
(a)



```
[01:41:05] Connected to: Buoyancy
[01:41:05] RX: *****-- Main Menu -- *****
[01:41:05] RX: #Number Option
[01:41:05] RX: #1 Set Mission Data
[01:41:05] RX: #2 Configure Vehicle
[01:41:05] RX: #3 Data Files
[01:41:05] RX: #4 Start Mission
[01:41:05] RX: #5 Go to Idle
[01:41:05] RX: #6 Go to Configure
[01:41:05] RX: #9 Reprint Main Menu

Device:   Buoyancy - ready
```

(b)

Figure 5.3: nRF UART v2.0 - **a:** Scan for devices **b:** Main Menu is displayed when connected to buoyancy vehicle.



BLE Keyboard
00:A0:50:00:00:08 (paired)

Buoyancy
DF:29:98:B4:B6:6F

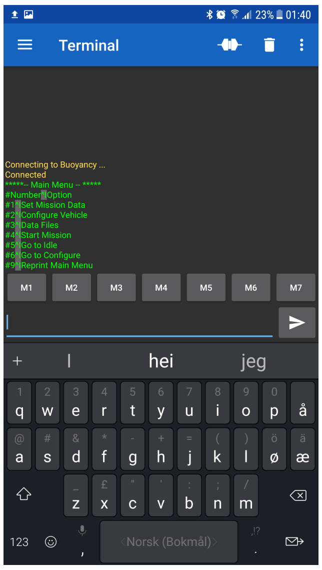
- unnamed -
24:88:52:4B:56:DC

- unnamed -
2E:55:01:88:CB:FE

- unnamed -
60:BC:D4:3A:F0:F2

- unnamed -
D4:6A:6A:6D:28:B4

(a)



```
Connecting to Buoyancy ...
Connected
*****-- Main Menu -- *****
#Number^IOption
#1^ISet Mission Data
#2^IConfigure Vehicle
#3^IData Files
#4^IStart Mission
#5^IGo to Idle
#6^IGo to Configure
#9^IReprint Main Menu
```

(b) connected to Buoyancy Vehicle and Main Menu is displayed

Figure 5.4: Serial Bluetooth Terminal - **a:** scan for devices **b:** Main Menu is displayed when connected to buoyancy vehicle.

5.3 Peripheral Drivers

A set of basic drivers is written as 'bare bone' drivers by following the nRF52832 datasheet. Initially all of these drivers are used to access peripherals, however as time progress it is found that creating peripheral driver instances from the SDK is better in terms of determinism and resource usage. While 'bare bones' library certainly work as intended the interrupt context approach is better in terms of code quality and execution overhead. The only driver remaining as 'bare bone' is the PWM driver, which control the RGB power LED. This is left 'bare bone' because it was made specifically to adjust RGB brightness and does not have to wait for a reply of any sort, so it won't introduce any significant execution overhead that interrupt context will alleviate.

5.3.1 PWM

The PWM peripheral driver is written as a 'bare bone' driver where registers values are manipulated without an abstraction layer on top. The three channels of the LED driver module each require a PWM signal to control each individual colored LED. The listing below show the configuration of channel 0, for RED LED. Each channel are configured exactly the same.

```c
int16_t REDseq_buf[] = {(1 << 15) | 0};  /**< Channel 0: Inverse polarity (bit 15),
                                                500us duty cycle. */

void pwm_init(void)
{
 // PWM0 RED pin
 NRF_GPIO->DIRSET = (1 << OUTPUT_PIN0);
 NRF_GPIO->OUTCLR = (1 << OUTPUT_PIN0);

 // PWM0
 NRF_PWM0->PRESCALER   = PWM_PRESCALER_PRESCALER_DIV_16; // 1 us
 NRF_PWM0->PSEL.OUT[0] = OUTPUT_PIN0;
 NRF_PWM0->MODE        = (PWM_MODE_UPDOWN_Up << PWM_MODE_UPDOWN_Pos);
 NRF_PWM0->DECODER     = (PWM_DECODER_LOAD_Common       << PWM_DECODER_LOAD_Pos) |
 (PWM_DECODER_MODE_RefreshCount << PWM_DECODER_MODE_Pos);
 NRF_PWM0->LOOP        = (PWM_LOOP_CNT_Disabled << PWM_LOOP_CNT_Pos);

 NRF_PWM0->COUNTERTOP = 2000; // 2ms period


 NRF_PWM0->SEQ[0].CNT=((sizeof(REDseq_buf)/sizeof(uint16_t))<< PWM_SEQ_CNT_CNT_Pos);
 NRF_PWM0->SEQ[0].ENDDELAY = 0;
 NRF_PWM0->SEQ[0].PTR = (uint32_t)&REDseq_buf[0];
 NRF_PWM0->SEQ[0].REFRESH = 0;
 NRF_PWM0->SHORTS = 0;

 NRF_PWM0->ENABLE = 1;
 NRF_PWM0->TASKS_SEQSTART[0] = 1;

 .
 .
 .
 updateLED(1, 1, 1); // Set minimum brightness
}
```

First a sequence buffer array is created, named REDseq_buf[] to hold the PWM counter sequence used when configuring the channel 0 sequence, and to later set the LED brightness. Then the PWM output pin is set to 'output' and deactivated. The next few lines are configuring the channel 0 to set the clock period to 1 μs, set the pin to OUTPUT_PIN0, set counter mode to up_MODE, the decoder is set to common load, and refresh counter. The LOOP setting is set to LOOP the sequence. And lastly the counter top is set to 2000, which make the the counter count 2000 μs (2 ms) which give the PWM frequency of 500 Hz specified by the datasheet for the LEDdriver[49, p. 10].

Next the PWM channel 0 sequence is configured by giving the length of the sequence buffer, setting end delay to 0, passing the address of sequence buffer to the channel sequence pointer, refresh the sequence to 0, and no short cut functions on events. Lastly the PWM channel is enabled and started.

When all the channels are configured a function for setting the brightness for switching in all three LEDs to minimum brightness of 1 %.

The listing below show the update led function.

```c
void updateLED(uint16_t REDduty, uint16_t GREENduty, uint16_t BLUEduty)
{
 NRF_PWM0->TASKS_STOP = 1;
 NRF_PWM1->TASKS_STOP = 1;
 NRF_PWM2->TASKS_STOP = 1;

 // Multiply dutycycle by 20 to make percenetage to appropriate compare value
 uint16_t REDcompare = REDduty*20;
 uint16_t GREENcompare = GREENduty*20;
 uint16_t BLUEcompare = BLUEduty*20;

 REDseq_buf[0]   = ((1 << 15) | REDcompare);
 GREENseq_buf[0] = ((1 << 15) | GREENcompare);
 BLUEseq_buf[0]  = ((1 << 15) | BLUEcompare);

 NRF_PWM0->SEQ[0].CNT = 1;
 NRF_PWM1->SEQ[0].CNT = 1;
 NRF_PWM2->SEQ[0].CNT = 1;

 NRF_PWM0->SEQ[0].PTR = (uint32_t)&REDseq_buf[0];
 NRF_PWM1->SEQ[0].PTR = (uint32_t)&GREENseq_buf[0];
 NRF_PWM2->SEQ[0].PTR = (uint32_t)&BLUEseq_buf[0];

 NRF_PWM0->TASKS_SEQSTART[0] = 1;
 NRF_PWM1->TASKS_SEQSTART[0] = 1;
 NRF_PWM2->TASKS_SEQSTART[0] = 1;
}
```

The function take in three 16 bit unsigned integer values that typically are values between 0 and 100. These are percentage values for LED brightness. RED, BLUE, and GREEN respectively. When changing sequence counter values the PWM channels have to be stopped, setting TASKS_STOP = 1 for each channel. A compare variable are defined for RED, BLUE, and GREEN PWM sequence channel where the input variables are multiplied by 20 to so that if input variable is 1, the respective channel sequence will be have a duty cycle of 20 μs 1 %, which give 1 % of maximum brightness. If input variable is 100 the duty cycle will be 2000 μs, 100 % LED brightness. The PWM channel counter is set to 1 to prepare start, the three channels is given its respective buffer address before the sequence start task is called by setting NRF_PWM->TASKS_SEQSTART[0]. = 1; on channel 0, 1, and 2 resepectively.

Two additionally functions are made to start, and to stop the PWM. And one function to test the LEDs where three for-loops to cycle through brightness strength in increments of 10 to loop through the color RGB color specter.

Detailed description of the PWM module and registers is found in the product specification 1,4 at: [20, p. 495-512].

5.3.2 TWIM

To interface sensors the I^2C/TWIM driver instance is created with the SDK driver, following the description in section 5.1.1. The driver instance is created by calling

nrfx_twim_t m_twim = NRFX_TWIM_INSTANCE(0);
An event handler is defined to handle the events defined by the driver description. One example is the NRFX_TWIM_EVT_DONE called when the TWIM event is done, either send event or receive event.

Send and receive function and created with the TWIM module macros:

- NRFX_TWIM_XFER_DESC_TX: Send byte to module/sensor address.
- NRFX_TWIM_XFER_DESC_RX: Receive byte for module/sensor address.
- NRFX_TWIM_XFER_DESC_TXRX: Send byte, then receive byte from module/sensor address
- NRFX_TWIM_XFER_DESC_TXTX: Send byte, then send another byte to module/sensor address

The full module description can be found here: [50]

5.3.3 UART

The UART driver is created to interface the stepper motor interface through a RS3232 module. The app_uart driver from the SDK is used to create the UART driver. It is configured in the uart_init() in main.c, by setting the respective RX and TX defined in main.h. RTS and CTX control pins are also set, though these are undefined as they are not used when flow control is disabled. Baudrate is set to 9600. The app_uart driver use TX and RX FIFO buffer. It is configured in uart_init() by calling the macro: APP_UART_FIFO_INIT. The macro takes in the uart initialization parameters defined above, the define TX and RX fifo buffer size, defined in main.h, and set the interrupt priority to high.

To send a command to the interface the specified 9 byte message format is used, shown in the listing below:

```
void sendCmd(uint8_t addr, uint8_t Cmd, uint8_t Type, uint8_t motor, long value)
{
 uint8_t txBuffer[9] = {0};

 txBuffer[0] = addr;
 txBuffer[1] = Cmd;
 txBuffer[2] = Type;
 txBuffer[3] = motor;
 txBuffer[4] = value >> 24;
 txBuffer[5] = value >> 16;
 txBuffer[6] = value >> 8;
 txBuffer[7] = value & 0xff;
 txBuffer[8] = 0;        // CRC calculated and added below

 for(int i = 0; i < 8; i++)    // Calculate CRC
 txBuffer[8] += txBuffer[i];
 for(int i = 0; i < 9; i++)    // Send message
 app_uart_put( txBuffer[i]);
}
```

- addr: motor address
- Cmd: command number

- Type: command type
- value » 24: Value - 8 Most significant bits
- value » 16: Value - 8 next bits
- value » 8: Value - 8 next bits
- value & 0xff: value Least significant bits
- CRC: The 8 previous byte values accumulated and sent as CRC value for error checking.

Afte the CRC is calculated and each byte starting with byte 0 is sent by calling app_uart_put().

To receive message from the motor interface an event handler is defined and keep track of the number bytes received. When 9 bytes are received the CRC is calculated. if CRC is equal to the 8 preceding bytes a function is called that put the received data in a global data structure in order to use it in the application.

Based on this driver a few functions are defined to send configuration commands, set piston position, set motor speed, read motor speed, and other status registers.

Detailed description of the app UART initialization and configuration is found in: [51]

5.3.4 SD Card and SPI

The SD card module driver also provide an SPI driver to interface the SD card.

To create the SD card module with SPI driver the SPI pins are define before calling the macro: NRF_BLOCK_DEV_SDC_DEFINE. The macro define the instance name, configures the SD card file system block size, and configure the SPI interface with the previously define pin numbers.

The SD card driver functions are based on the SDK provided FATFS example. It provide an example where SD card disk is initialized, opened, files are listed, and a new file is created and written to before the disk is unmounted.

Several new function are defined to count and list all files starting with integer numbers with the .txt extension, create new file, and read from file, to mention a few. Detailed description of the SD card example is found in: [52] Detailed description of the FATFS driver is found in: [53]

5.3.5 SAADC

The SAADC module is, as described in section 3,3,1, a hardware peripheral used to measure voltage from the pressure sensor output, and battery voltage. The SAADC peripheral driver is in it self quite simple. It consists of the initialization function saadc_init() and an event handler. Two analog input pins are defined for its respective SAADC channel. Channel 0 for measuring battery voltage, and channel 1 for measuring pressure sensor output voltage.

The configuration settings are listed in following listing:

```
void saadc_init()
{
 ret_code_t err_code;

 // Default SAADC configuration
 nrfx_saadc_config_t saadcConfig;

 // SAADC channel configuration
 nrf_saadc_channel_config_t batteryChannelConfig, pressureChannelConfig;


 saadcConfig.resolution = NRFX_SAADC_CONFIG_RESOLUTION;
 saadcConfig.oversample = NRFX_SAADC_CONFIG_OVERSAMPLE;
 saadcConfig.low_power_mode = NRFX_SAADC_CONFIG_LP_MODE;
 saadcConfig.interrupt_priority = NRFX_SAADC_CONFIG_IRQ_PRIORITY;

 // Battery Measurement
 batteryChannelConfig.resistor_p = NRF_SAADC_RESISTOR_DISABLED;
 batteryChannelConfig.resistor_n = NRF_SAADC_RESISTOR_DISABLED;
 batteryChannelConfig.gain       = NRF_SAADC_GAIN1_5;
 batteryChannelConfig.reference  = NRF_SAADC_REFERENCE_INTERNAL;
 batteryChannelConfig.acq_time   = NRF_SAADC_ACQTIME_40US;
 batteryChannelConfig.mode       = NRF_SAADC_MODE_SINGLE_ENDED;
 batteryChannelConfig.burst      = NRF_SAADC_BURST_ENABLED;
 batteryChannelConfig.pin_p      = (nrf_saadc_input_t)(NRF_SAADC_INPUT_AIN0);
 batteryChannelConfig.pin_n      = NRF_SAADC_INPUT_DISABLED;

 // Pressure Measurement
 pressureChannelConfig.resistor_p = NRF_SAADC_RESISTOR_DISABLED;
 pressureChannelConfig.resistor_n = NRF_SAADC_RESISTOR_DISABLED;
 pressureChannelConfig.gain       = NRF_SAADC_GAIN1_5;
 pressureChannelConfig.reference  = NRF_SAADC_REFERENCE_INTERNAL;
 pressureChannelConfig.acq_time   = NRF_SAADC_ACQTIME_40US;
 pressureChannelConfig.mode       = NRF_SAADC_MODE_SINGLE_ENDED;
 pressureChannelConfig.burst      = NRF_SAADC_BURST_ENABLED;
 pressureChannelConfig.pin_p      = (nrf_saadc_input_t)(NRF_SAADC_INPUT_AIN1);
 pressureChannelConfig.pin_n      = NRF_SAADC_INPUT_DISABLED;
 .
 .
 .
}
```

The four lines of saadConfig are default configurations for the SAADC module. It sets common configuration values according to settings in the config.h file. The Following confiurations are set in config.h:

- Resolution: 14 bit
- Oversampling: 64 times
- Low Power mode: 0 (disabled)
- Interrupt priority: 6 (0 to 7 where 0 is highest)

The next configuration values are define for channel 0 and channel 1, battery and pressure sensor respectively. both channels have equal configuration settings, but are separated in case of future changes. Gain is set to $\frac{1}{5}$ in order to have the highest measured value of 3,0 V be equal to the highest internal reference value of 0,6 V. Acquisition time is 40 μs, and the channel is set to 'single ended mode' as suppose to differential mode. Burst mode is enabled to allow for oversampling on more than one channel. As the mode is set to 'single ended mode' only the positive 'p' channel input is enabled, for AIN0, while negative channel 'n' is disabled.

The reset of the initialization function are functions for setting configuration to channels, and for setting SAADC buffers that hold the sampled values to be copied to application variables.

The event handler are mainly used to notify the application when the SAADC sampled values are ready to be read. On an NRFX_SAADC_EVT_DONE is ready the sampled values are copied to the mission data structure, and the SAADCdataReady flag is set to true to notify the application. Every 10 samples the SAADC module is calibrated to for temperature changes.

Detailed description of the SAADC driver is found in: [54]

5.4 PID Regulator

The PID regulator is implemented with the PID controller library created by Rubén Santa Anna, at Geek Factory [55]. The library is intended to fit any microcontroller with support for floating point operation. It comes with example code specifically targeted PIC microcontrollers. To fit this library in with the nRF52832 the lines `tick_get();` and is replaced with `app_timer_cnt_get();` and `#define TICK_SECOND 32768UL` is placed in PID.h to use the Real Time Counter (RTC) instead of systick. The library is written with inspiration from the creator of the official arduino PID library with through explanation of this is found at [56].

5.5 Finite State Machine

The core program flow is managed by a Finite State Machine (FSM). State diagram with transition conditions are illustrated in Figure 5.5. It is structured with five main states, INIT, IDLE, CONFIGURE, MISSION, and PICKUP, and three additional states: SLEEP, FAILURE, and LOWPOWER. The later mentioned serve critical support roles for when the normal program flow is either encounter an exception, such as error handling or critically low battery, the system is not being operated and need to be put to sleep to save power. Below follows a brief description of each state:

- **RESET:** Activated RESET button trigger a hard reset. The core modules and peripherals are initiated before the FSM is entered.

- **INIT:** First state entered when system is reset by RESET button or a soft reset from system off mode, triggered by hall-effect button. The INIT state start a timer for sampling SAADC every 10 second to measure battery, and a timer to updating FSM every 500 ms in order be responsive to change while it can be put to system-On sleep and respond when events and interrupts occur. In addition the various fsm struct flags are initialized to false. The SAADC sampled variables, battery and pressure is populated with default values, LOW_POWER_THRESHOLD+10 (+10) in order to avoid triggered LOWPOWER state before first measurements is ready while still maintain a predictable initial value. The pressure variable is initiated with the minimum SAADC value that correspond to 0.5 V from pressure sensor, which is expected value when the sensor is not in the water. After initialization the FSM transitions directly to IDLE state.

- **IDLE:** The IDLE state a serve as a middle state between operational and system-off mode. It is not advertising BLE, but provide by activating the hall-effect button it switches to CONFIGURE state which advertises BLE. On the other hand it starts a one-shot timer that counts up to 2 minutes, and if not stopped (still in IDLE state) within 2 minutes the system enter system-off mode. The IDLE state also measure batter power and handles errors, so that if battery power is at threshold or below it will interrupt operation before a mission is configured or started. The same goes for errors. The state is indicated by minimum birghtness on the RGB LED, shown as a dim white light.

- **CONFIGURE:** The LED change color to pure blue to indicate the configure state. BLE is advertising and can be connected to by the dedicated desktop terminal application "BuoyancyApp", or by a serial terminal application on a phone/tablet. Default settings for two missions are pre-configured so that if no specific configuration is need the BLE application is not needed. In most situations custom configuration is necessary. Next state, MISSION, is triggered either by hall-effect button or by BLE-command "#4 Start Mission". The BLE-command "#5 Go to Idle" will transition the state machine back to IDLE state and a new 2 minute timer is started to put it to sleep if inactive.

- **MISSION:** When mission starts the BLE connection is terminated by server, the LED switches to green to be visible in the water, and mission is run according to Figure 5.8. The function `prepareMission()` calculate the number of configured mission, and create a new mission Log file to record the mission. When final mission is finished or aborted (by hall-effect button) the new piston position is set to 0.0 to make sure the vehicle float to surface. Transition to next state, PickUp state, is triggered when the final mission is finished, or by activating hall-effect button.

- **PICKUP:** When piston reaches position 0.0, or bottom end limit switch and velocity is zero the LED change to yellow to signal PickUp state. BLE is advertising in order to re-configure the vehicle, transfer files, delete files or transition to another state. IDLE, CONFIGURE, and MISSION state can be entered through BLE-command. If hall-effect button is activated the FSM transitions to IDLE state. That way the hall effect button can be used to all the actively operate states without the need for BLE application. when IDLE state is entered the BLE conection is terminated and advertising is stopped.

- **SLEEP:** The SLEEP state is entered when the timer in IDLE state reach 2 minutes. The SD card is unmounted and system-off mode is entere by calling `sleep\_mode\_enter()`, this function is called by the FSM module in order to access the function: `sd_power_system_off()` in main module. The system-off mode is the minimum power mode and wake-up require hard reset, (activate RESET button) or activate the hall-effect button, which trigger a soft reset. All settings will then be initialized to default upon reset.

- **LOWPOWER:** Battery power is measured ever 500 second during mission, and every 10 second otherwise. In the case that voltage fall below 12,8 V the LOWPOWER state is entered directly, without a regular state transition. This is done because it is highly critical that motor has time to set piston to 0.0, corresponding to 0 meter, or float to surface. Otherwise the vehicle may be lost. The LED is changed to bright RED to signal LOWPOWER state, and SD card is unmounted. Then system is entering a while loop that that call __WFE() to put system in system-on sleep mode. Hard reset is required upon entering LOWPOWER state.

- **FAILURE:** In the case of a detected fault in software the failure state is entered directly to avoid the fault to surface as unpredictable behavior during operation. Mission is aborted and vehicle is set to float to surface by setting piston position to 0.0. SD card in unmounted, LED is changed to PINK to signal FAILURE - fault handling - and the nordic SDK error handler takes care of the detected fault. The system require hard reset upon entering FAILURE state.

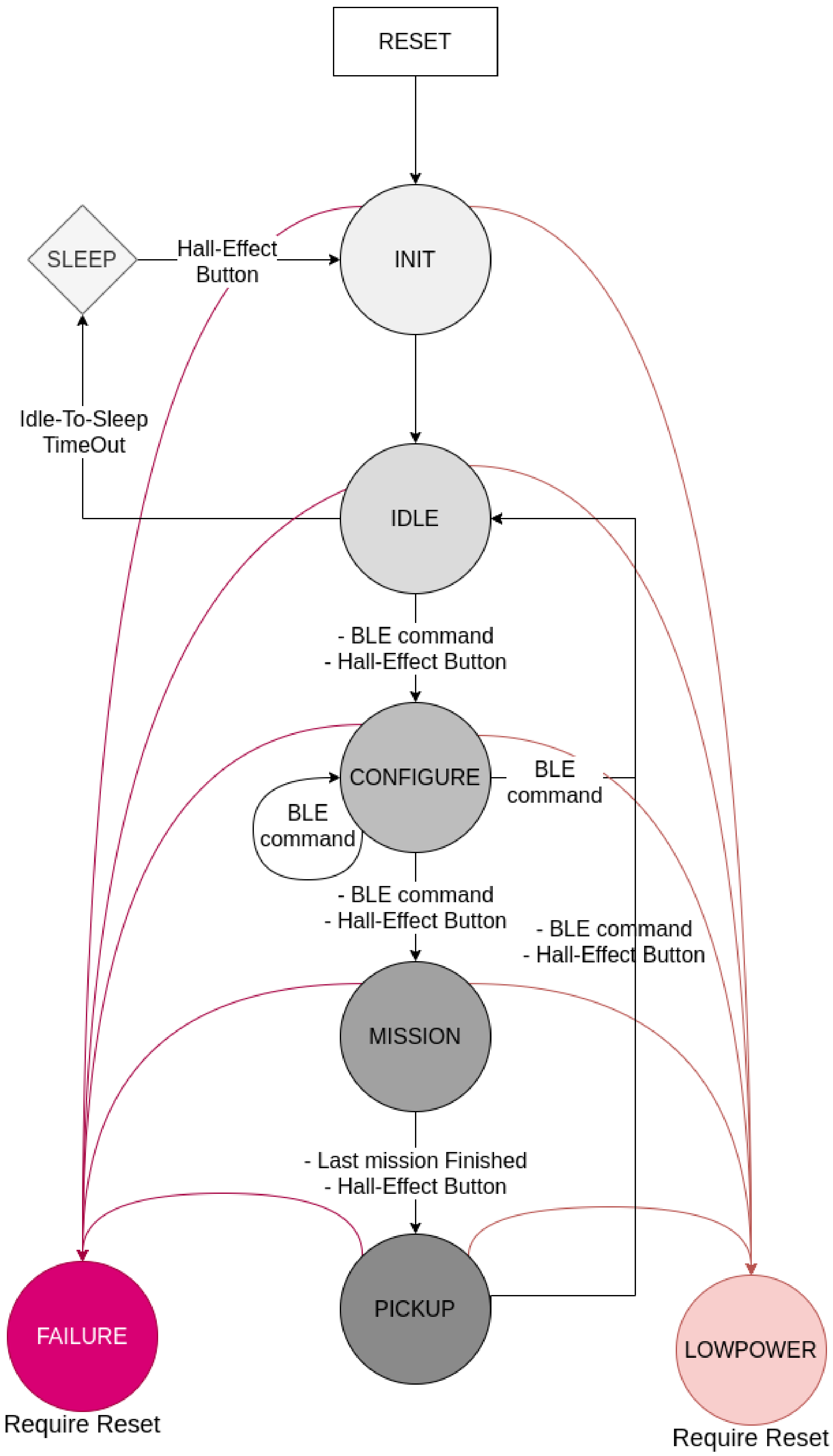


Figure 5.5: State diagram to illustrate the program flow. The state transitions are triggered by Hall effect button and/or BLE command input. In the case of detected error the FAILURE state is entered, and if low battery power is detected the LOWPOWER state is entered, both exception states require a hard reset (reset button activated).

5.6 Menu

Figure 5.6 is a block diagram to give an overview of the user interface menu tree structure. Main Menu is presented to client when connection is established. The operator at client side choose menu options with integer numbers 0-9, where 0 or any non-integer values 'cancel' the operation.

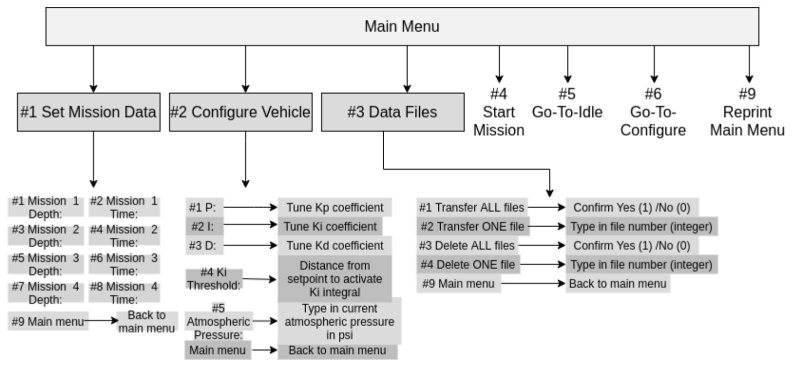
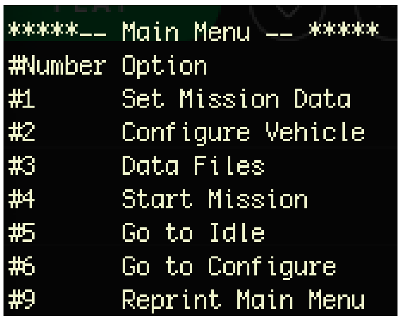


Figure 5.6: Diagram to illustrate the user interface menu structure. From Main Menu the submenus; Set Mission Data, Configure Vehicle, and Data Files are entered to read and/or set configuration options, or perform file operations. The other Main Menu options are are state transitions to: MISSION, IDLE, and CONFIGURE.

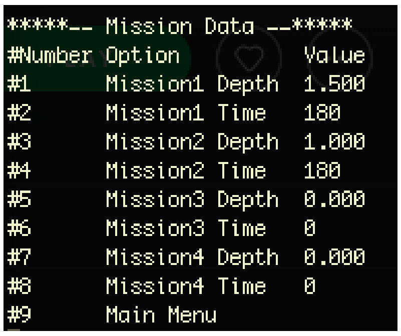
Figure 5.7 show the Main Menu window, the Configure Vehicle menu and an example of a successful file transmission. When in Main Menu the first option is: "Set Mission Data" where a total of 4 missions can be configured with depth set point in meters (decimal number) and time in seconds (integer number). Number 9 is pressed to go back to Main Menu. The next menu option is "Configure Vehicle" where PID Controller coefficients are tuned (decimal numbers), Ki threshold is set (decimal number), and atmospheric pressure compensation is set in psi (decimal number). 9 is pressed to return to Main Menu.

The third menu option is to handle SD card data files. All the stored mission log files are printed as a list with name and size in number of bytes. The first option is to transfer all mission log files. A confirmation message is printed where user has to press 1 to confirm, or 0 to cancel. Any other value than 1 will cancel the operation. The second file operation option is to transfer a single file from the printed list. A message is printed to screen telling the user to chose a file to transfer, or press 0 to cancel. When transmission start the name of the file is printed to screen. During transmission the number of bytes transferred are continuously updated to show progress. When transmission is ended an end status report is printed to screen that display the filename, size, transmission time, and transmission speed in bytes per second ($\frac{B}{Sec}$).



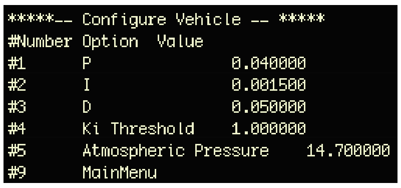
```
*****-- Main Menu -- *****
#Number Option
#1      Set Mission Data
#2      Configure Vehicle
#3      Data Files
#4      Start Mission
#5      Go to Idle
#6      Go to Configure
#9      Reprint Main Menu
```

(a)



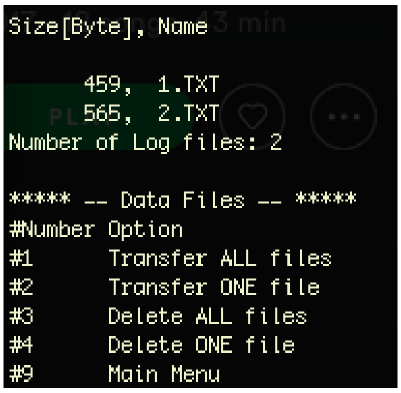
```
*****-- Mission Data --*****
#Number Option         Value
#1      Mission1 Depth 1.500
#2      Mission1 Time  180
#3      Mission2 Depth 1.000
#4      Mission2 Time  180
#5      Mission3 Depth 0.000
#6      Mission3 Time  0
#7      Mission4 Depth 0.000
#8      Mission4 Time  0
#9      Main Menu
```

(b)



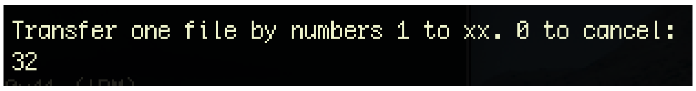
```
*****-- Configure Vehicle -- *****
#Number Option  Value
#1      P               0.040000
#2      I               0.001500
#3      D               0.050000
#4      Ki Threshold    1.000000
#5      Atmospheric Pressure    14.700000
#9      MainMenu
```

(c)



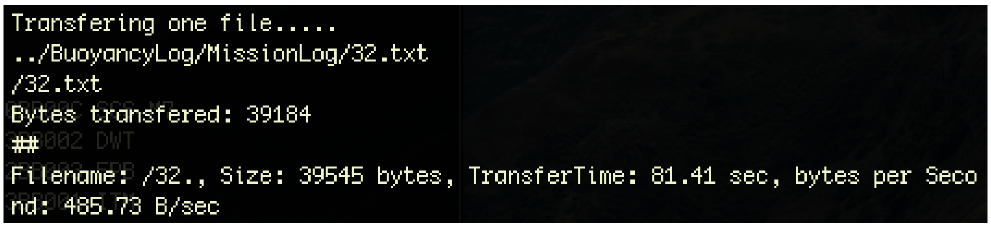
```
Size[Byte], Name

      459,  1.TXT
      565,  2.TXT
Number of Log files: 2

***** -- Data Files -- *****
#Number Option
#1      Transfer ALL files
#2      Transfer ONE file
#3      Delete ALL files
#4      Delete ONE file
#9      Main Menu
```

(d)



```
Transfer one file by numbers 1 to xx. 0 to cancel:
32
```

(e)



```
Transfering one file.....
../BuoyancyLog/MissionLog/32.txt
/32.txt
Bytes transfered: 39184
##
Filename: /32., Size: 39545 bytes, TransferTime: 81.41 sec, bytes per Seco
nd: 485.73 B/sec
```

(f)

Figure 5.7: Terminal application - **a:** Main Menu **b:** Mission Data menu **c:** Configure Vehicle Menu **d:** Data Files menu **e:** Choose file to transfer **f:** Status message for finished file transmission.

The third file operation option is to delete all files. The same confirmation message as for transfer all file are displayed. Press 1 to confirm deletion of all files, or 0 to cancel - where any other key than 1 will cancel operation. When operation finish an end status report is printed with the number of files deleted and the number of files that failed to delete. The fourth file operation option is to delete a single file. The same message as for transferring one file is printed to screen where user have to choose a file to transfer, or 0 to cancel. If the file was successfully deleted a success message is printed, otherwise a fail message is printed. 9 is pressed to go back to Main Menu. In Main Menu 4 is pressed to start a new mission. 5 is pressed to go to Idle state, where the vehicle will go to sleep after 2 minutes of inactivity. 6 is pressed to go to configure mode. this is used when in PickUp mode in order to go to configuration mode. When in configuration mode there is no action taken when 6 i pressed. 9 is pressed to reprint Main Menu.

5.7 Timers

Most timer are using the application timer (app_ timer) driver from the Nordic SDK. It run off of the RTC[57] which is a 24 bit counter running off of the 32,678 KHz Low Frequency Clock (LFCLK)[58]. the timer has only 24 bit it will overflow after:

$$max\ periods = 2^{24} = 16\ 777\ 216\ periods$$

$$maximum\ number\ of\ minutes = \frac{16\ 777\ 216\ periods}{32\ 678\ \frac{periods}{s} \cdot 60 \frac{s}{min}} \approx 8,56\ min$$

and as the maximum time recommended used with the app_ timer is half the oveflow time[59], the maximum time to use with the application timer is 4,28 min. The RTC timer is energy efficient in that no additional hardware need to be utilized and consume powered. The application can be used as long as the timer value is less than 4,28 minutes.

The following timers are used with the application timer:

- Update FSM timer: Make sure the FSM is handled and updated at least every 500 ms, other wise the system appear unresponsive.
- Sleep timer: When in Idle mode the sleep timer is started counting to 2 minutes. If the system is still in Idle mode and the timer is not stopped the system will go to sleep mode to save power.
- Motor stop timer: The timer is set when all missions have ended and motor is set to float to surface. The timer count 5 seconds before the system is allowed to handle stop motor in case of an interrupt from the bottom limit switch.
- Measure battery timer: When not running in mission mode the battery is measured and evaluated if above minimum threshold. The timer trigger a sample every 10 second.
- SampleSensorData timer: When running in mission mode the SAADC and TMP117 are sampled every 500 ms.
- UpdateMissionLog timer: A timer that trigger a snapshot of mission data values to the mission log. When the snapshot is taken the mission program will proceed and a flag is set to write missionLog data to SD card.

The only hardware timer used is the mission timer. The mission timer is updated to count to the number of seconds specified in each individual mission.

It runs off of the timer1 peripheral, a 32 bit counter configured to run at 125 kHz. Following the same calculation as above to find the maximum number of minutes for a single mission.

$$max\ periods = 2^{32} = 4\ 294\ 967\ 296\ periods$$

$$maximumnumberof minutes = \frac{16\ 777\ 216\ periods}{125\ 000\ \frac{periods}{s} \cdot 60\ \frac{s}{min}} \approx 572,66\, min \approx 9,54\, hrs$$

with the recommended half time: $\frac{572{,}66\ min}{2} = 286{,}33$ min – roughly 4 hrs and 46 min.

5.8 Digital Filter

As analog measured data is susceptible to noise a digital low-pass filter is implemented to counteract rapid changes, as a result from noise, in the measured pressure signal. As the vehicle is moving relatively slowly in the water the filter can have a low cut-off frequency, in other words a high dampening effect effectively smoothing the input pressure signal.

An Exponential Moving Average (EMA) filter is implemented[60]. EMA essentially save the state of a measured value and use it along with a filter coefficient EMA$_\alpha$ to calculate to calculate the filter response effect on the next sampled value. EMA$_\alpha$ is a value ranging from 0 to 1. High EMA$_\alpha$ has a slow response to rapid changes in pressure, and take many samples into account. Higher EMA$_\alpha$ has a faster response, higher dampening effect to rapid changes, and take fewer samples into account. EMA$_\alpha$ can therefor be though of as an inversely related cutoff frequency. Higher EMA$_\alpha$ yield lower cutoff frequency, as a result of the faster response.

`#define EMA_alpha 0.1` is the defined EMA$_\alpha$ coefficient.
`float EMA_state = 0.0;` initialize the EMA state variable, which is save previous measured and filtered pressure value. The filter calculation is placed as the first line in the function `CalcPressureAndDepth()` as such:

```
EMA_state = (EMA_alpha*mission.MeasuredData.pressure) + ((1-EMA_alpha)*EMA_state);
```

The EMA$_\alpha$ is multiplied with the measured pressure data where the previous measured data is multiplied by the (1 - EMA$_\alpha$).[60]

5.9 Mission

At system boot the mission module is initialized by setting 'mission' and 'pid' structures to 0, then populate them with relevant default data values, example default PID coefficients, depth and time for mission 1 and 2, while mission 3 and 4 is set to 0. Then the PID instance is created and configured with direction, output limits, sample time, and initially set to manual. Manual refer to disabled, while auto refer to enabled controller.

Figure 5.8 illustrate the mission program flow. Before mission starts the function `prepareMission()` is run to set the latest PID tuning coefficients and calculate the number of missions to be run. Number of missions is calculated by condition that for a given missionNr time is different from NULL.
The mission log is initiated by setting the data structure to 0, then a new mission log is created and made ready to write the mission log. Finally the 10 second battery measurement timer is stopped as battery will be measured along with pressure every 500 ms during mission.

Mission is started by calling `runMission()`. A timer to update the mission log once a second is started, and an initial timestamp is taken to mark the start of the mission execution at time 0.0 ms.
To fetch and execute the number of configured missions a for-loop with index, i, started at 0 and continue execution until it is equal to the number of missions. the for-loop encapsulate the entire mission execution and i is increased each time a mission is finished.
On the beginning of each iteration a missionId is given equal to the current number of finished missions. and the missionId variable is used to populate the current mission with target depth and time. This is shown is the listing below:

```
uint8_t missionId = mission.missionFinished;
mission.currentMission = mission.missionNr[missionId];
```

Before the actual mission can start the mission timer compare value is updated with with current mission time value, before it is started. In addition the PID and SAADC sampe timers are started.
The core mission code section is encapsulated within a while loop that execute as long as missionId is equal to the number og finished missions, as such:

```
while(missionId == mission.missionFinished){
```

Everything within the while loop is conditionally executed, as illustrated in Figure 5.8. The follow boolean flags are used for execution within the while-loop:

- sampleSAADC: Flag to signal SAADC need to sample battery and pressure
- SAADCdataReady: Flag to signal SAADC data is ready to be read
- calculatePID: Flag to signal new PID calculation.
- missionLogUpdated: Flag to signal mission log is updated
- TMP117dataReady: Flag to signal TMP117 data is ready to be read
- fsm.hallEffectButton Flag to signal hall-effect button activated

The sampleSAADC timer set the sampleSAADC flag to true. The if statement is then evaluated to true, and both SAADC and TMP117 sensor is sampled.

When SAADC has finished sampling it sets SAADCdataReady = true. The condition statement evaluating SAADCdataReady is then evaluated to true, and the pressure data is used to calculate pressure represented in voltage, psi, and pascal. Psi pressure is then used in calculate depth in meters.

Current position is read from the motor controller to write it to mission log. Then PID is used to calculate new piston position based on measured depth in meters. When PID has finished a timestamp is taken, before the new piston position is sent to motor controller, directly based on PID output value.

As it is critical that buoyancy vehicle have enough power to float to the surface before battery is completely empty the measured battery power is continuously compared to the threshold voltage: `LOW_POWER_THRESHOLD = 12.8`. In any case the measured voltage is less or equal to threshold all timers are stopped, mission log file is closed, and LOWPOWER state is entered.

`__WFE()` is called to enter system-On mode, which is a light sleep state, which it is waken up from at any event. Then if hall-effect button is triggered this sets the flag in fsm structure, and the conditional is evaluated to true, so that mission is aborted. This is a handy functionality as sometimes the hall-effect butten is triggered by accident if a magnet comes too close the sensor. It would be a waste of time to have to wait for mission to finish in the case where a, or several, mission was configured.

When mission timer handler is triggered the missionFinished value is incremented, which make the while loop evaluate to false. The mission relevant timers are then stopped to make ready for a updated timer value, before new mission is started, if configured. Otherwise all missions are finished. The Motor is then set to float the vehicle to the surface, and a timer is set to measure motor velocity. This trigger the motorstop() function when velocity reach 0, in order to put motor to sleep and save power. The timer is need to add a delay in order for the motor start and gain velocity.

Finally the current mission log file is closed, and timer for measuring battery every 10 seconds is restarted to keep track of battery voltage.

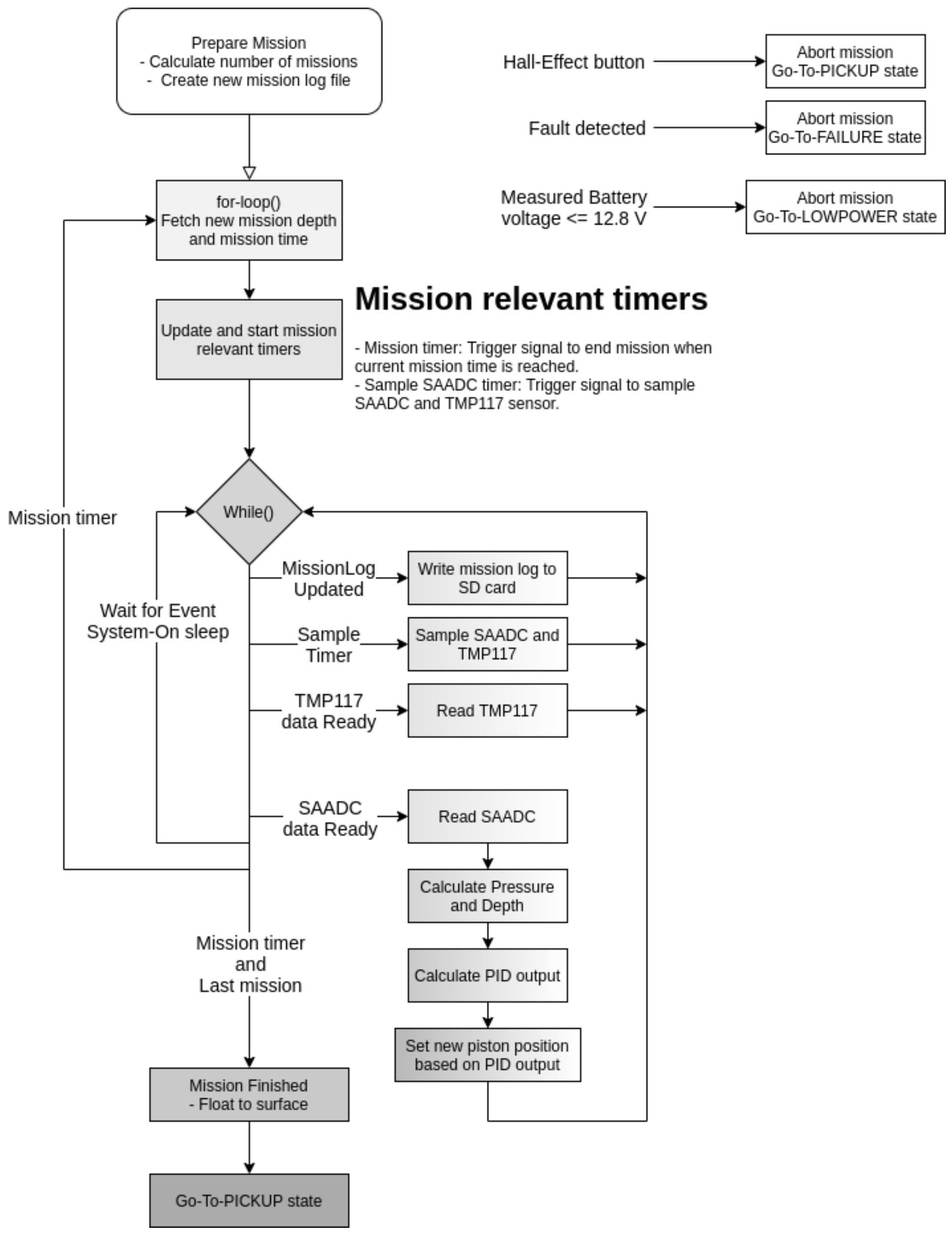

Prepare Mission
- Calculate number of missions
- Create new mission log file
for-loop()
Fetch new mission depth and mission time
Update and start mission relevant timers
While()
Mission timer
Wait for Event System-On sleep
MissionLog Updated
Write mission log to SD card
Sample Timer
Sample SAADC and TMP117
TMP117 data Ready
Read TMP117
SAADC data Ready
Read SAADC
Calculate Pressure and Depth
Calculate PID output
Set new piston position based on PID output
Mission timer and Last mission
Mission Finished
- Float to surface
Go-To-PICKUP state
Hall-Effect button
Abort mission Go-To-PICKUP state
Fault detected
Abort mission Go-To-FAILURE state
Measured Battery voltage <= 12.8 V
Abort mission Go-To-LOWPOWER state
Mission relevant timers
- Mission timer: Trigger signal to end mission when current mission time is reached.
- Sample SAADC timer: Trigger signal to sample SAADC and TMP117 sensor.

Figure 5.8: Block diagram to illustrate mission program flow; A function is called to make initial calculations before the mission starts, the a for-loop is entered to run all the configured missions where each mission is running inside a while loop until a timer-based interrupt increment a variable that breaks the loop.

5.10 Motor Configuration

The stepper motor interface is configured configured by sending commands and setting corresponding axis parameter values. Message sequence is explained is section 5.1.3 under the UART driver description.

5.10.1 Motor Initialization

The motor interface is initialized by calling the function `motorInit()`. It configures the interface with a set of key settings, the following list provide an overview of the configuration commands and values set to initialize the motor interface:

- **Enable limit switches:** Before running the motor it is important to enable the two limit switches, upper and bottom. When limit witches are enabled the motor will not run in the direction towards the switch in case of a failed switch connection. If, however, the switch is not enabled the motor won't recognize the switch or the mechanical barrier. Failling to stop the motor before reaching a mechanical limit result in risk of breaking the motor or one or more physical structures of the vehicle.

- **Set maximum motor current:** Running the motor for extended length of time on too much current will lead to overheating and breakage of the motor components. Excess current consumption also drain the battery fast. Setting motor current is divided in 32 steps with setting parameter values from 0-255. The maximum current is limited to 700 mA by writing value 60 to axis parameter nr. 6 [Maximum Current].

- **Set standby current:** When motor is not running a standby current can be set to reduce the total current consumption and generated heat. the maximum current is divided in 256 steps ranging from 0,0 A to 2,8 A. The standby current is set to 0,0 A by writing value 0 to axis parameter 7 [Standby Current].

- **Power Down Delay:** After motor has stopped there is a time delay until standby current is entered. The standby current delay is set in multiples of 10 ms by writing values ranging from 0 to 65535. The standby current delay is set to 1000 ms by writing value 100 to axis parameter 214.

- **Freewheeling:** The coil current can be cut off entirely setting the motor in freewheeling state. The time delay is set in the same way as the power down delay for standby current in multiples of 10 ms by writing values ranging from 0 to 65535. The Freewheeling delay is set to 5 seconds by writing value 500 to axis parameter 204.

- **Maximum positioning Speed:** This configuration set the maximum motor speed when moving towards a specified position. It ranges from 1 to 2047. The maximum velocity is set to 1024, according to calculations shown in subsection 5.10.2 below, by writing value 1024 to axis parameter 4.

- **Maximum Speed in velocity mode:** This configuration set the maximum motor speed when rotating in a left-or-right direction, not towards a specified position. It ranges from 1 to 2047. The maximum velocity is set to 1024, according to calculations shown in subsection 5.10.2 below, by writing value 1024 to axis parameter 2.

- **Pulse Divisor:** The pulse divisor is a translation unit from the motor interfaces internal velocity to real world velocity. It is explained in further detail in subsection 5.10.2 below. The pulse divisor is set to 1, by writing value 1 to axis parameter 154, according to calculations in subsection 5.10.2 below.

- **Set position to zero:** Finally the internal position counter is set to zero to set the reference point for max vehicle density when the piston is positioned all the way to the bottom limit switch. The piston counter is set to zero by writing value 0 to axis parameter 1 [Actual Position].

5.10.2 Velocity

This section describe how to calculate, and configure, motor- and piston velocity, based on instructions in PD-1161 TMCL^{TM} Firmware Manual, available in [32, p. 90 - 91]

The piston velocity is defined by the motor shaft rotational velocity. The motor use internal units v_{int} (int values -2048 to 2048) with a pulse-divider ranging from 1 to 13 to translate internal units to real world velocity units. pulses/microsteps per second (pps). The following formula is used to convert internal units to pps units:

$$v_{pps} = \frac{16 \cdot 10^6 \cdot v_{int}}{2^{pulse\_div} \cdot 2048 \cdot 32}$$

The pulse_ div can be read or written on axis parameter #154. and v_{int} is the internal velocity unit, which can read and/or written with the following axis parameters: #2, #3, #4, #130, #181, #182, #194, #195.

In order to convert pps to real world velocity the following equation is used to calculate to rounds-per-second:

$$v_{rps} = \frac{v_{pps}}{r_{fullstep} \cdot r_{microstep}}$$

and from rounds-per-second to rounds-per-minutes one simply multiply with 60:

$$v_{rpm} = v_{rps} \cdot 60$$

The following symbols are used:

- v_{rps}: Velocity in rounds-per-second
- v_{rpm}: Velocity in rounds-per-minute
- v_{pps}: Velocity in internal unit pulses/microsteps -per-second
- $r_{fullstep}$: Fullstep resolution, the number of fullsteps per full revolution. For mosts motors, including the one used in this project: 200
- $r_{microstep}$: Microstep setting, the number of microsteps per full step (default: 256) Can be read of written from/to by axis parameter #140
- *rev*: revolution - one revolution is a 360° rotation.
- *fullstep*: motor has 200 fullsteps on a 360° rotation.
- *pulse/microstep*: Each fullstep is divided in a configurable number of microsteps. In this case 256 microsteps per fullstep. *Pulse* is an alias for microstep.

The following calculations are done to set the piston velocity to 2,44 $\frac{mm}{s}$.
The lead screw has 1 mm threads, which will move the piston head 1 mm per revolution. The motor must therefor be configured to a velocity of 2,44 rps.
with default $r_{microstep}$: 256 the internal velocity can be found by:

$$v_{pps} = v_{rps} \cdot r_{fullstep} \cdot r_{microstep}$$

$$v_{pps} = 2,44 \cdot 200 \cdot 256 \left[\frac{rev}{s} \cdot \frac{fullstep}{rev} \cdot \frac{microstep}{fullstep} \right]$$

$$v_{pps} = 123\ 928 \frac{microstep(pulse)}{s}$$

and choosing pulse_ div: 1 The internal velocity v$_{int}$ is found:

$$v_{int} = \frac{v_{pps} \cdot 2^{pulse\_div} \cdot 2048 \cdot 32}{16 \cdot 10^6}$$

$$v_{int} = \frac{123928 \cdot 2^1 \cdot 2048 \cdot 32}{16 \cdot 10^6} = 1015,21$$

By rounding up v_{int} to 1024 $\frac{steps}{s}$ the final piston velocity is: $2,4414 \frac{rev}{s}$

5.10.3 Torque and Maximum Depth

According to Børseth's[3] calcultions on piston velocity and torque the maximal depth, with respect to gauge pressure, experienced by the piston is 80,7 m. This is with piston velocity 2,44 $\frac{mm}{s}$ and 12 W motor power. This give a safety margin of almost 30 meters, $\frac{2}{3}$ the maximal specified depth of 50 meter.

With the new motor configuration settings the maximum motor current is set to 700 mA, which yield input power of:

$$P_{mot} = V \cdot I_{max} = 12.8V \cdot 700mA = 8.96W.$$

$\frac{2}{3}$ the power in børseth's calculations. Assuming linear relation in change of motor power the maximum depth is down to 60,25 meter, which give safety margin of less than 10 meters. Safety margin of up towards 10 meters is good to have in case something were to pull the vehicle deeper than 50 meters + overshoot.

Chapter 6

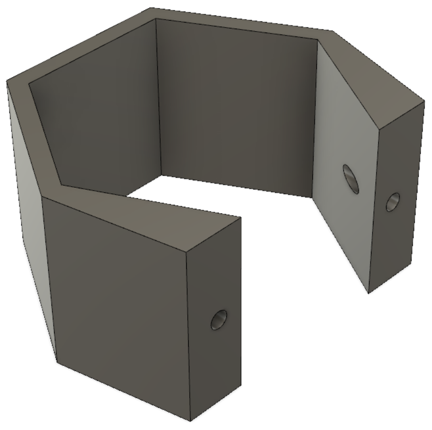
Mechanical Modifications

To satisfy the requirements derived from the use cases several mechanical modifications are necessary; The TMP117 module is mounted to the pressure sensor brass nut to achieve best possible temperature measurement of the surrounding water. Limit switches are mounted to the linear actuator to be triggered when the piston head is at the bottom or top of the piston. The Hall-Effect sensor is placed inside the vehicle House in order to control the state machine with a magnet. Cables are made to connect sensors and switches to the PCB on top of the inner lid. A protective cylindrical cover is designed to avoid wires from getting trapped between lead screw of linear actuator and the limit switch holders. A section of inner lid is cut out to make enough room to handle the wiring coming up from the vehicle House. Finally an LED housing is designed and machined to place the indication LED on top of the vehicle. The LED housing serve both as a transparent housing to make the indication LED visible in all stages of vehicle operation, as an air venting hole to prevent air to compress during assembly and outer lid adjustment, and hole to access the physical reset button in case it is needed to perform a hard reset.

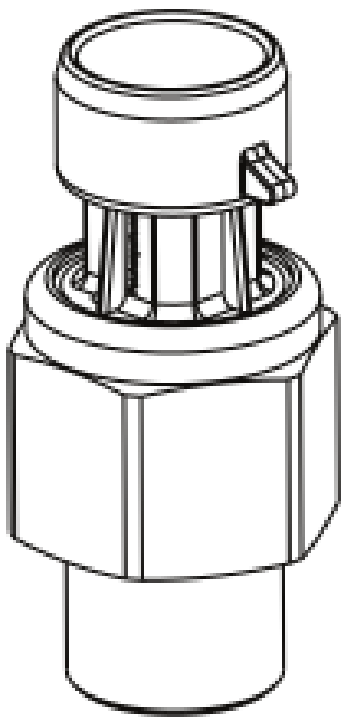
The mechanical modifications have been 3D-modeled and machined in collaboration with the mechanical workshop "mekanisk verksted" at the Department of Engineering Cybernetics.

6.1 TMP117 Holder

The 3D printed TMP117 holder, Figure 6.1a, is designed to be pressed down on the brass nut of the PX3 pressure sensor, Figure 6.1b. The brass not is screwed in through the bottom of the vehicle house for the piezo resistive pressure sensor to get in contact with the water. Brass is the best thermal conductor for the temperature sensor to measure outside water temperature. The TMP117 module is mounted on to the holder by two screws.



(a) TMP117 holder to place the TMP117 sensor on the PX3 pressure sensor brass body

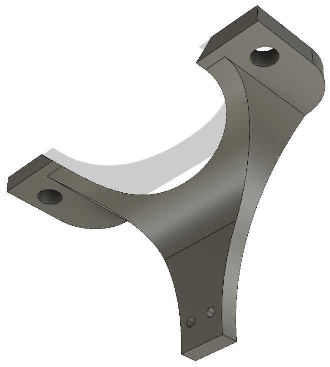


(b) PX3 brass nut on which the TMP117 sensor holder is mounted. Reproduced from datasheet [28]

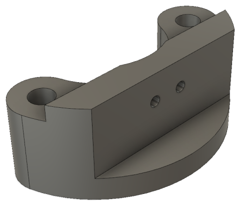
Figure 6.1: TMP117 holder and PX3 pressure sensor brass nut.

6.2 Limit Switch Holder

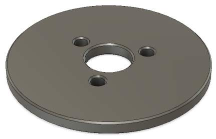
The two limit switches that restrict the motor from exceeding the pistons physical limits are held by two 3D printed holders. The Holder for upper limit switch, Figure 6.2a, is fastened on two of the four M4 bolts that hold the motor to the inner lid. The switch is fastened to the holder with two small screws. The bottom switch holder, Figure 6.2b is placed and fastened on top of the housing cap, seen in Figure 1.2b. The switch is fastened in the holder with two small screws. The switches are triggered by a circular metal slice, as in Figure 6.2c, fastened by three screws on top of the linear actuator holster, see Figure 1.2b. The metal slice on top of the holster trigger a limit switch when the piston is getting close to one of the piston limits. The limit switches reduces the maximal distance which the piston head can travel from 57 mm, to 52.02 mm. The piston position from the piston bottom is measured to 1.25 mm, while the maximal piston position is 53.27 mm.



(a) Top limit switch holder, mounted on two M4 bolts underneath the stepper motor block



(b) Bottom limit switch holder mounted on top of the piston housing cap



(c) Circular metal slice to trigger limit switches when placed on top of the leaded holster in the linear actuator

Figure 6.2: Upper and bottom end limit switch holders, and circular metal slice to trigger the switches

6.3 Hall Effect Sensor Placement

Figure 6.3 illustrate where in the Vehicle House the Hall effect sensor is placed. Its placement is somewhat inconvenient as a mark can't be placed exactly where the magnet should be held to trigger the sensor, since the sensor is placed vertically so far up that it is on the inside of the rotated Outer Lid. However a mark is placed at the bottom outside of the Vehicle House just below the sensor location, so that once the magnet is used to operate the vehicle a few times it is easy to know where to place the magnet to trigger the sensor. The sensor is placed 12 cm up from the bottom of the vehicle, at the same angular position where the wires come up through the inner lid to the PCB.

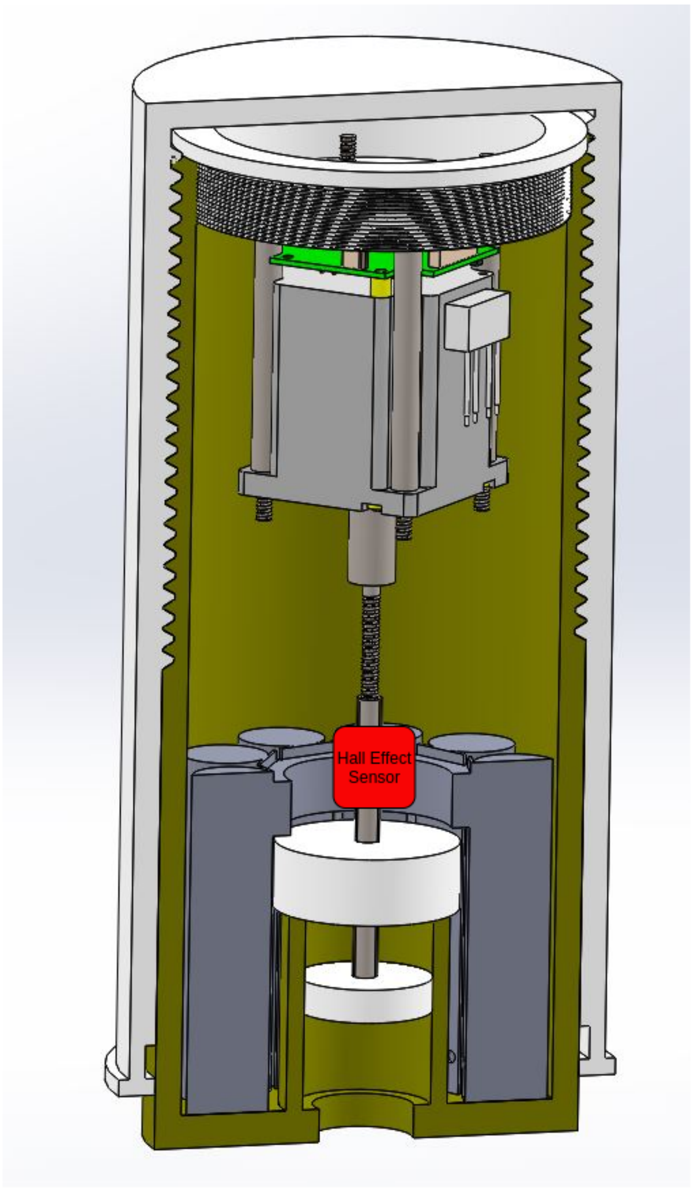


Figure 6.3: Illustration of where in the vehicle House hall-effect sensor is placed

6.4 Protective Cylinder Cover

The added wires from sensors below the stepper motor and the added limit switch holders cause problems when the vehicle is taken apart. When, for example, batteries are being changed the Inner Lid is screwed off, causing the motor block and the upper limit holder to rotate. This happens as the wires from under the motor are unplugged from the PCB to allow inner lid to come off. The rotating motor block with the limit switch holder catch the non-rotating wires and to the point where they get tangled and inner lid get stuck, or one of the limit switches and holder are damaged. To avoid this from happening a cylinder cover, attached to two of the M4 screws holding the stepper motor to the Inner Lid (the two other than that hold the upper limit switch holder), is designed and 3D printed.

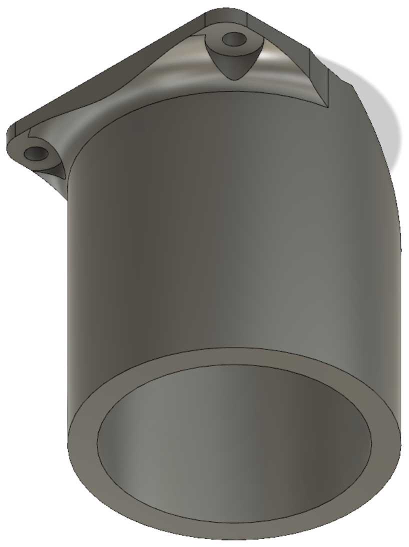


Figure 6.4: Protective Cylinder Cover to avoid wires from getting tangled in the lead screw of linear actuator and limit switch holders, mounted on two M4 bolts underneath the stepper motor block

6.5 Machined Inner Lid

As wires from the battery, px3 pressure sensor, TMP117 sensor, Hall effect sensor, and both upper and bottom limit switches need to be connected to the PCB, a bigger hole is needed to pull the wires up through inner lid, and connect them to the PCB. Figure 6.5 show the shape of the section that is machined out of the Inne Lid to make space to handle the wires. The section machined out is jus a little smaller than the shape of the PCB to avoid the PCB to potentially be damaged by wires being pulled up and down.

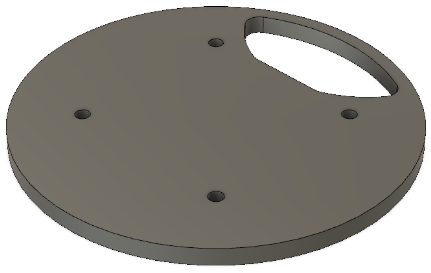
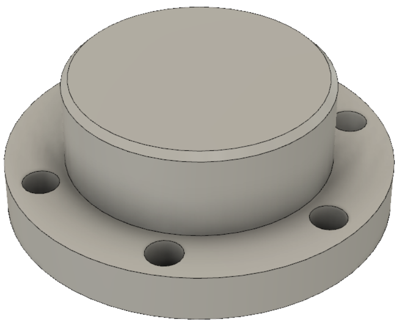


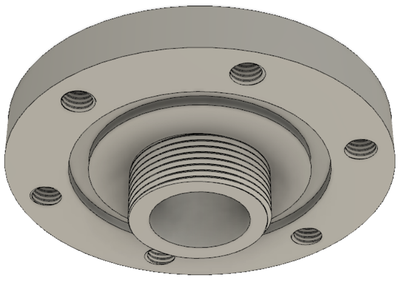
Figure 6.5: Show the removed section of inner lid to provide access for wiring to the PCB

6.6 LED Housing

The LED housing serve multiple purposes. First and foremost it is a housing for the power LED that indicate the state which the program vehicle program is in, and to make it visible in water during mission – and PickUp mode. To make the housing transparent, the housing top shown in Figure 6.6a, is machined out of one piece of acrylic Plexiglas. The bottom piece, Figure 6.6b, is machined from one piece of rigid PVC. It has a threaded tube to be able to screw it onto a threaded hole on top of the Outer Lid. It has six threaded screw holes in order to make a tight fit to the top piece. an O-ring is placed between the top and bottom piece to seal it from water leaks. An O-ring is also placed between the bottom piece and the outer lid to seal from water leaks. As previously mentioned the LED housing serve as a vent to avoid air to compress inside the vehicle is assembled, or the Outer Lid position is adjusted. This is important to prevent the gauged pressure sensor from measuring pressure based on compressed air pressure inside the vehicle. The LED house is screwed on to the Outer Lid after the Outer Lid is adjusted, to maintain atmospheric pressure inside the vehicle. Another benefit that comes with the LED housing is that during assembly of the vehicle the LED itself is not yet connected, as this would lead to a number of rotations on the cable, which would cause strain on LED, PCB, and cable itself. Instead the 45 cm long cable is thread through the outer lid hole while the outer lid is assembled and adjusted. The LED housing is connected to the LED cable and screwed on to the Outer Lid. As there is some extra room for the LED and cables, the LED is loose enough to move freely. As the LED housing is rotated the LED is not, and thereby avoid the cable inside the vehicle from being twisted. The high power LED tend to generate a lot of heat when operated with high currents. The heat sink in Figure 6.6c is placed underneath the LED to help dissipate heat more effectively.

The LED housing increase the total vehicle volume with $4.9106 \cdot 10^{-5} m^3$


(a) 3D-model finished designed LED housing top, 6 bolts to hold top and bottom together with o-ring in between for waterproofing


(b) 3D-model finished designed LED housing bottom, 6 bolts to hold top and bottom together with o-ring in between for waterproofing, o-ring between bottom and outer lid, and a threaded cylindrical tube for mounting to inner lid and provide wiring to LED

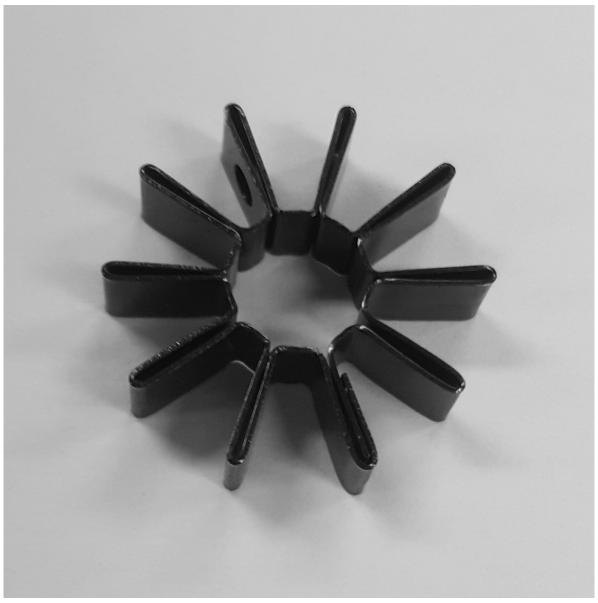

(c) Heatsink placed under power LED inside the LED housing to create better heat dissipation

Figure 6.6: A first draft concept for a transparent view of the LED housing

6.7 Adjusting Outer Lid position with Added Weight

The added components increases the vehicle's mass by 141 g from 6.101 kg to 6.242 kg. The LED housing minus the reduced piston head position range increases the vehicle volume with $5.06768 \cdot 10^{-5}\ m^3$ at minimum piston head position, and $5.37933 \cdot 10^{-5}\ m^3$ at maximal piston head position.

The total vehicle volume and density is adjusted by the Outer Lid, and is measured from the bottom of the Vehicle House to the bottom of Outer Lid. The reduced piston head position range also reduces the density range at a given Outer Lid position.

Table 6.1: Outer Lid position to adjust vehicle's density range (ρ_{min} to ρ_{max}). $h_{add_{bot}}$ is the distance from the Vehicle House bottom to the Outer Lid bottom. *Rotations* is the number of full rotations + additional rotation, to reach the given $h_{add_{bot}}$ position.

$\rho_{min}[\frac{kg}{m^3}]$	$\rho_{max}[\frac{kg}{m^3}]$	$h_{add_{bot}}[cm]$	Rotations
992.0	1003	9.26	7 + 82°
994.0	1005	9.17	7 + 140°
996.0	1007	9.07	7 + 196°
998.0	1009	8.98	7 + 254°
1000.0	1011	8.88	7 + 310°
1004.0	1015	8.68	8 + 73°
1008.0	1020	8.49	8 + 188°
1012.0	1023	8.33	8 + 280°
1016.0	1027	8.16	9 + 26°
1020.0	1031	7.97	9 + 137°

Table 6.1 shows the Outer Lid position table from børseth's thesis[3, p. 36], adjusted for the added mass, volume, and piston head position range.

The density is calculated with Equation 2.3, where volume is the summation of all the different contributing elements, minus the volume related to minimum and maximal piston head position.

The table is expanded with a few extra rows to provide more details for the density values from 992 $\frac{kg}{m^3}$ to 1000 $\frac{kg}{m^3}$.

The minimum and maximum Outer Lid position $h_{add_{bot}}$ is:

- minimum: 3,7 cm
- maximum: 13,0 cm

A graph showing the density range as a function of the Outer Lid position range is included in Appendix E.

Chapter 7

Test and Verification

The nature of product development is to circle back and forth between implementation and re-evaluation. Therefor a lot of testing has been done through the development phase. However, though individual parts - mechanical, electrical, and software/firmware - are found to perform well on its own, assembling the parts to a complete system can cause unexpected behavior. Examples may be unpredicted heat from electrical component, due to less air circulation within a closed container, or timing issues in software due to process interruptions from added modules. Most of these cases are taken into account during planning and design stages. However there are always details forgotten, or are simply not thought of. To best possible alleviate system failure a test system need to be put in place where the functioning of each individual part are tested, and evaluated whether its performance satisfy requirements.

7.1 Functional Test Plan

A test plan is made to carry out functional verification before the system is ready to be tested on water. The test plan is intend to test all the operational aspects of the vehicle, such as Hall effect button, indication LED, state transitions, mission Log files, etc. The functional test plan is included in Appendix F

7.2 Current Consumption

A simple current consumption measurement is made, to be able to compare with design estimations in subsection 3.3.10, though power consumption is the more correct term for measurements [61]. The final assembled system is measured with both revision 1 and 2 of the PCB. The current is measured while powered by a 24 V power supply, using a TENMA 72-2605 multimeter, and is measured with and without the motor connected. The following are tested:

- Sleep Mode
- Idle Mode

- Configure Advertise Mode
- Configure connected Mode
- Motor Running constant
- Motor Regulate average
- Transferring data file
- Low Power state

7.3 BLE Configuration Effect On File Transmission

As discussed in subsection 5.2.1 BLE is configured for optimized transmission rates. Tests are made to explore the effects of connection interval and client device negotiation values on file transmission rate.
The tests are performed with the outer lid taken off to create best possible signal strength. A 39,155 kB file is transferred to the BuoyancyApp on a MacBook Air, running OSX Catalina, and Serial Bluetooth terminal app on a Samsung Galaxy S6, running Android.

The first test is made with connection interval set to 20 ms. The second test is made with connection interval set to 7,5 ms. The Negotiated Data length and MTU size is reported and presented. As Samsung desire to negotiate MTU size to 512 bytes, an additional test is made where max MTU Size is set to 512 and max GAP data length is set to 251.

7.4 Test In Water

The vehicle is tested in three separate locations; A freshwater tank, a freshwater lake, and at a pier for test in salt water.

7.4.1 Freshwater Tank

The first location is a 1,64 m deep freshwater tank at the Department of Electronic Systems. The purpose of this test is to check for the vehicles floating capability with the added volume and weight, test the functionality in water, and on PID controller tuning. In order to precisely know the water density, a CTD (conductivity, temperature, and Depth) instrument is used to measure depth and density. Several tests were made to test and tune the PID controller. The piston velocity is set to 1 $\frac{mm}{s}$.

Three key test cases are highlighted:

1. - P: 0.02
   - I: 0.00
   - D: 0.00
   - Mission 1: 1.5 m

- Mission 2: 2.0 m
- Mission 1 and 2 time: 180 sec

2.
   - P: 0,02
   - I: 0,01
   - D: 0,01
   - Mission 1: 1.3 m
   - Mission 2: 1.0 m
   - Mission 1 and 2 time: 180 sec

3.
   - P: 0,02
   - I: 0,01
   - D: 0,05
   - Mission 1: 1.3 m
   - Mission 2: 1.0 m
   - Mission 1 and 2 time: 180 sec

7.4.2 Freshwater Lake

The second location was originally planned to be carried out at the large-scale sea cage, Aquatrass. As this is not an available option at the time of testing, the second test is performed in a freshwater lake. It has a concrete pier going out to where the water is 2,3 meter. The water is still, with a minimal current flow, and well suited for early outdoor testing. A density profile is not made as the CTD instrument is not available at the time of testing. Adjustments for outer lid and PID tuning is thus made based on trial and error. A fishing line is tied to the LED housing for safety to mitigate risk of losing the vehicle in case an error occur. Test focus on PID tuning, where configuration settings are tested on various depths and logged data area analyzed to find the best possible tuning parameters. The test environment is significantly different than the original location at the fish farm facility. The lake has significantly less variation in density than the ocean, which is deeper, has stronger currents, and is affected by salinity. In addition waves on the ocean could be a benefit in PID tuning to test the robustness. On the other hand, tuning the PID is a simpler task as the conditions are more likely to remain constant throughout the experiment. based on experience from test in the water tank, with results presented in section 8.2, piston velocity is increased from 1,0 $\frac{mm}{s}$ to 2,44 $\frac{mm}{s}$. The added velocity provide increased PID controller bandwidth, which makes the system more responsive to the regulators output values.

7.4.3 Salt Water

The vehicle is briefly tested at a floating pier, in sea water, to test how the vehicle behave in salt water. The test is done with the PID parameters obtained from test in the freshwater lake. CTD instrument is not available, therefor no density-profile is made for the given test.

The vehicle is configured with the following values:

- P: 0.04
- I: 0.01
- D: 0.07
- Atmospheric pressure: 14.7 psi

- Mission 1: 1 m
- Mission 2: 2 m
- Mission 1 and 2 time: 180 sec

7.5 Wireless Connection In Water

To test capabilities and limitations of wireless connection in and around water a series of tests are conducted. Four sets of tests are made, two in a fresh water environment, and two in salt water. At each location one set is done with a chip antenna, the other with a FlexPIFA™ antenna. signal strength is measured by RSSI with the nRF Connect Bluetooth developer application on a Samsung S6 mobile phone.

Test Setup

1. First the signal strength is measured and logged by measuring the Received Signal Strength Indicator (RSSI) on land at the location to see if the surrounding environment (fresh water and salt water) have an effect on the signal quality in free air. RSSI measurements are taken at a distance from 0,25 m to 16 m from the vehicle by doubling the distance on each measurement(0,25 m, 0,5 m, 1 m, 2 m, 4 m, 8 m, 16 m).

2. The second test is measuring RSSI with the vehicle floating in the surface, it normally float 2 cm beneath the surface, with the LED housing top out of the water. The measurements are taken at normal with respect to the surface, just above the vehicle. RSSI is measured at distance from 0,25 m to 2 m above the surface. Distance is limited to maximal reachable height above the floating vehicle.

3. The third test is where the vehicle is submerged beneath the surface. The depth is measured and RSSI is measured and logged to find the maximal depth where RSSI is measurable.

4. The fourth test is with the vehicle submerged under water and the desktop application is used to transfer a 40 kilo Byte (kB) data file. The transmission rate is measured and logged on each centimeter under the surface, until the

application is unable to finish the full transmission.

5. The fifth test intend to check the effect of connecting to the vehicle from an angle. According to theory presented in section 2.5 the angle between the antennas have an impact on the signal quality. The desktop application is used to check how far a distance and at what angle the connection is maintained for. Then a 40 kB data file is transferred to check the data rate at the maximal angled distance.

Chapter 8

Results and Discussion

This chapter presents the test results and a discussion of these. The Chapter is structured such that the combined result and discussion of each test are presented in the order that the tests were described in the previous chapter. Some of the implemented elements are presented and discussed as well, such as mechanical modifications and SoC capacity utilization.

8.1 On Land Test and Verification

This section presents the results and discussion a discussion of these for topics not directly related to subsurface activity. This include SoC utilization, mechanical modifications, PCB revision 1 transistors, functional test on land, current measurements, and data transmission rate.

8.1.1 SoC Capacity Utilization

The final design take up 20 of the total 31 GPIO pins, which account for 64,5 %. Figure 8.1 is a screen shot from Segger Embedded Studio that show the nRF52832 SoC Flash and RAM utilization. Used Flash: 322,9 kB of 512 kB (63 %), used RAM: 33,5 kB of 64 kB (52 %). There is enough room for further development without the risk of having to move to a new module with better specifications, which require a full redesign of both PCB and firmware. The design is also not over-engineered, which would be the case if only a fraction of the chips specifications were used, with the added cost, complexity, and PCB footprint.

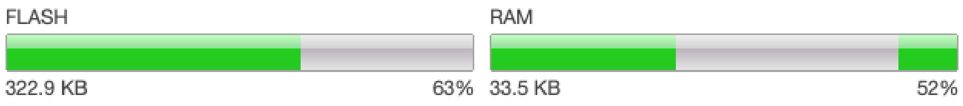


Figure 8.1: SoC Flash and RAM utilization. Flash: 63 % RAM: 52 %

8.1.2 Mechanical Modifications

Figure 8.2 show the LED housing with LED indicating the state modes: IDLE, CONFIGURATION, MISSION, Pick-Up, SLEEP, FAILURE, and LOW-POWER

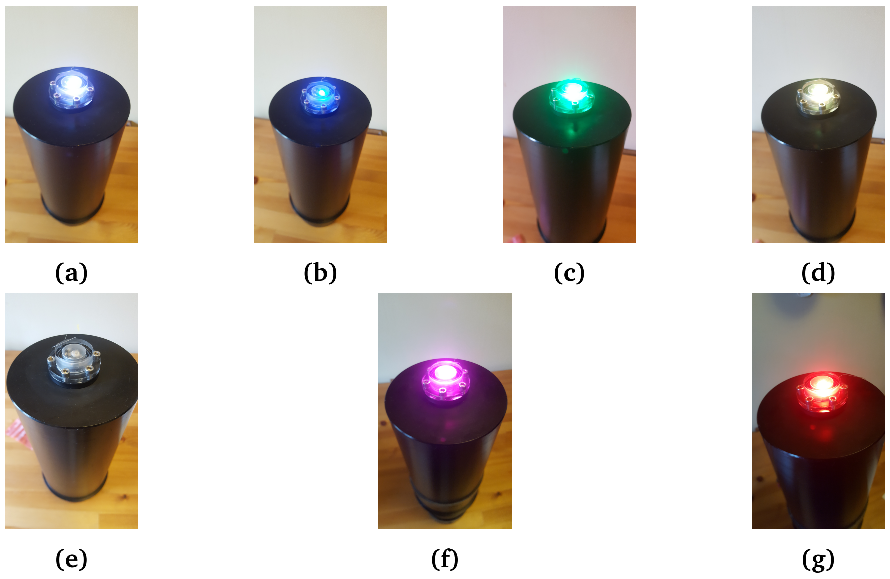
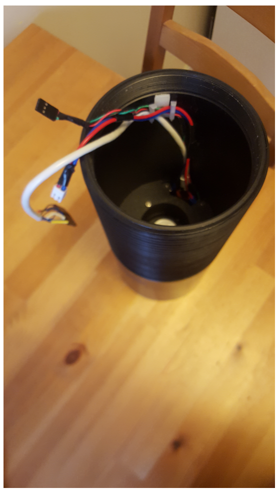
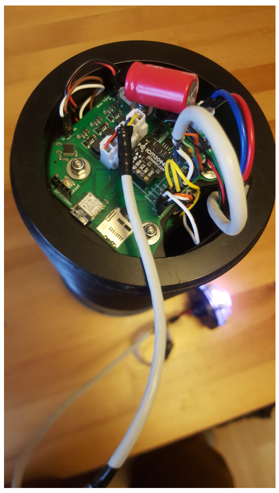


Figure 8.2: LED indicate current state - a: White IDLE b: Blue CONFIGURATION c:Green MISSION d: Yellow Pick-Up e: Off SLEEP f: Pink FAILURE g: Red LOW-POWER

Figure 8.3a show the wires coming up from under the linear actuator up through the inner lid to connect sensors and bottom limit switch to PCB. Figure 8.3b show the PCB with wires connected. In the middle is wires from PCB to LED housing to power the indication LED.



(a)



(b)

Figure 8.3: Wiring - a: Wires from Hall-Effect sensor, TMP117 sensor, Battery, and bottom limit switch b: Wiring from PCB to LED housing to power indication LED.

Figure 8.4 show the inner lid, stepper motor and protective cover. The stepper motor and cover is wrapped in tape to help smooth sharp edges in order to further mitigate risk of wires and connectors being hooked and tangled.

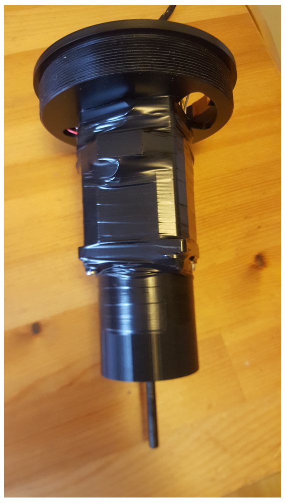
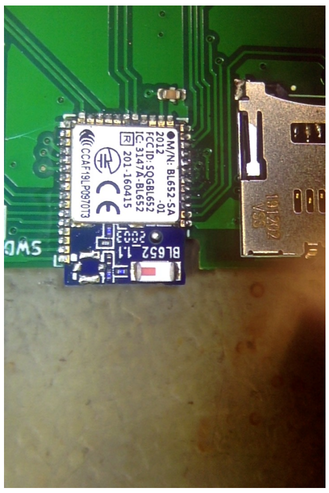


Figure 8.4: Protective cover over limit switch holder and linear actuator. Reinforced with tape to smooth sharp edges, mitigate risk of wires and connectors to get tangled

Figure 8.5a and b show the PCB where BL652 radio module is fitted with chip antenna and FlexPIFA$^{\text{TM}}$ antenna respectively. The FlexPIFA$^{\text{TM}}$ is mounted to the side of inner lid, above the section cut out for wires.


(a)

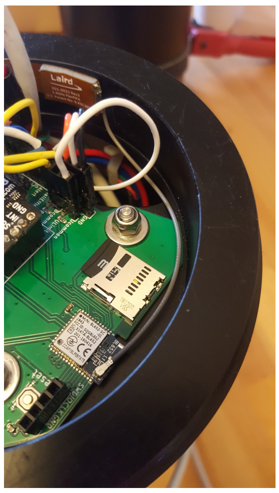

(b)

Figure 8.5: Antenna Configuration - **a:** PCB with ceramic chip antenna **b:** PCB with FlexPIFA™ antenna

8.1.3 PCB Revision 1 Transistors

The two transistors in PCB revision 1, intended to switch on/off power for PX3 pressure sensor and battery measurement, does not work as intended. In case of PX3 sensor transistor 2,5 V is measured on both emitter and collector side. The reason is that transistors are place at the wrong side of the circuitry. Instead of switch on/off positive node, it need to switch on/off GND node.

This is explained in the case of the PX3 sensor. The active emitter voltage is 5 V, equal to collector 5 V supply voltage, while base voltage is never higher than microcontroller pin output at 3.3 V. As base voltage transition from inactive (0 V) to active (3,3 V) current starts to flow. The collector -and emitter voltage are approaching equal. As emitter approach collector at 5 V, the emitter create a negative base-emitter voltage. This in turn cause a decreased base current flow to a certain point when transistor is no longer in saturation, as required base-emitter voltage is 0.95 V for saturation[62]. Collector, emitter, and base voltage stabilize at a certain voltage level.
The result is a voltage of approximately 2,5 V on each terminal. The solution is to place the transistor between sensor ground-pin and GND. As the transistor is turned off, the current flow through the sensor is cut off. with only a small leakage current. When base voltage reach above 0.95 V the transistor is in saturation and current is freely flowing through the transistor and sensor.

The PX3 transistor is changed as described above in PCB revision 2, presented in section 4.5, while the transistor intended to switch battery measurement is removed.

8.1.4 Functional Test Plan

The test plan in Appendix F is followed where all points are verified with successful behavior, and without errors. The magnetic activated Hall effect button is used to wake the vehicle up from sleep mode, and to transition from Idle mode to Configure mode, where the vehicle is broadcasting advertisement to establish a connection to user device. The various procedures are follow to verify successfully started and finished missions, with the correct number of generated mission log files.

8.1.5 Current Consumption

Table 8.1: Current consumption PCB revision 1.

PCB Revision 1:	Motor connected	Motor Disconnected
Sleep Mode:	63.7 mA	8.56 mA
Idle Mode:	67.8 mA	14.24 mA
Configure Advertise Mode:	65.7 mA	10.24 mA
Configure connected Mode:	65.7 mA	10.19 mA
Motor Running constant:	293 mA	N/A
Motor Regulate average:	215 mA	N/A
Transferring data file:	70.4 mA	13.24 mA
Low Power state:	96.5 mA	28.65 mA

Table 8.2: Current consumption PCB revision 2.

PCB Revision 1:	Motor connected	Motor Disconnected
Sleep Mode:	61.8 mA	6.19 mA
Idle Mode:	66.03 mA	10.4 mA
Configure Advertise Mode:	64.2 mA	8.39 mA
Configure connected Mode:	64.2 mA	8.34 mA
Motor Running constant:	292 mA	N/A
Motor Regulate average:	215 mA	N/A
Transferring data file:	67.1 mA	11.3 mA
Low Power state:	94.6 mA	26.64 mA

Table 8.1 and Table 8.2 show the measured current consumption of the system running off of PCB revision 1 and PCB revision 2, respectively. PCB revision 2 draw approximately 2 mA less current across all values. Except for when motor is active. This indicate that the reduction in current drawn is related to all states when motor is inactive, which is also when the pressure sensor is shut down by a transistor. According to datasheet the current consumed by the pressure sensor is 3,5 mA [28].

According to Table 3.4 the calculated passive current can go as low as 1,661 mA. This is under optimal configurations, and voltage dividers are not taken into account. A thorougly voltage and current measurements should be performed to find and minimize significant contributors to current consumption. There is a high chance that unconfigured sensors draw a fair amount of current as they may not be put in sleep mode by default.

From disconnecting the stepper motor and controller it is clear that it has a current consumption of approximately 55 mA when not running a mission. Even if sensors and microcontroller were highly optimized for low power consumption the controller interface still consume a lot. A solution may be to cut off the stepper motor power with a transistor when not configured or running mission. The transistor need to be able to handle the 4 A peak current and 30 V, the maximum motor voltage. Until a physical solution is made to reduce the current drain, the battery power need to be disconnected when the vehicle is not used for one hour or more. If not, the battery is drained which is negative for both environmental and economical reasons.

Idle state has higher current consumption than configuration state. This is surprising as radio is inactive. However, in idle state all three LEDs are used, while in configure state only the blue LED is lit, which is likely the reason. Current is increasing significantly during a data file transmission. This is because both the radio and SD card is active. It was expected that SD card would draw more current, since worst case estimation is as high as 100 mA in Table 3.4. A 15 - 20 mA is likely more realistic for estimated worst case. However in this case it seem to be barely 2 mA.

In low-power state the red LED is set to 10 % which is likely the cause of the high current consumption even with motor disconnected.

8.1.6 BLE Configuration Test and File Transmission Rate

Connection Interval: 20 ms

MacBook Air OSX:

- Negotiated Data lenght: 101 bytes
- Negotiated MTU size: 104 bytes
- Average Transmission rate: 622 B/sec

Samsung Galaxy S6 Android:

- Negotiated Data lenght: 101 bytes
- Negotiated MTU size: 104 bytes
- Average Transmission rate: 1586 B/sec

Connection Interval: 7,5 ms

MacBook Air OSX:

- Negotiated Data lenght: 101 bytes
- Negotiated MTU size: 104 bytes
- Average Transmission rate: 4096 B/sec

Samsung Galaxy S6 Android:

- Negotiated Data lenght: 101 bytes
- Negotiated MTU size: 104 bytes
- Average Transmission rate: 2112 B/sec

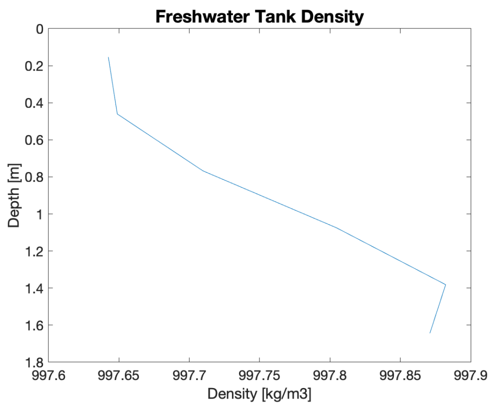
The results presented above show the clear benefit of minimizing the connection interval to improve transmission rate. The effect is especially evident on OSX that show an improved transmission rate of 658,5 %. The Android device has an improvement of 33,2 %
An important point to mention is that the mobile applications are printing out the file as transmission is ongoing. This create potential for a bottle neck, and is likely the explanation for the stagnation in transmission rate at 2112 B/sec at 7,5 ms, even though the negotiated values remain the same. A more interesting point to look is the transmission rate difference between OSX and Android devices at 20 ms, where Android device is 2,55 times faster than the OSX device. This is the case even as negotiated values are equal and the android has the bottle neck as it prints out all the transferred text to screen. It is unclear why this is the case. Further investigation is needed with the help of a BLE packet sniffer that display how packets are sent over the air, and compare how data is handled in code - primarily the vehicle code.

8.2 Freshwater Tank

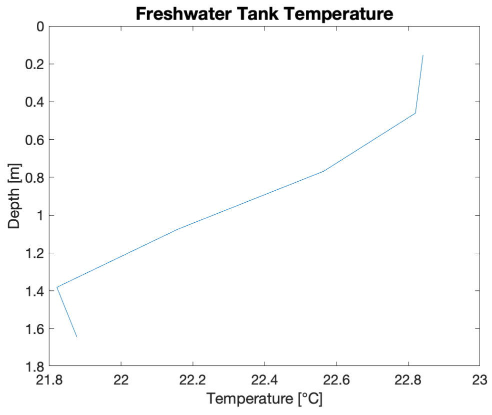
This section present and discuss results from test in freshwater tank. Outer lid adjustment is presented first, followed by a discussion of observed errors, and finally the test results from three sets of PID parameters.

8.2.1 Floating and Outer Lid Adjustment

The result of the CTD density measurement (calculated by CTD instrument from measured pressure, temperature and salinity) is presented as the density profile in Figure 8.6a, and the temperature in Figure 8.6b. The chlorinated fresh water density is roughly 997,65 $\frac{kg}{m^3}$ at the surface and 997,87 $\frac{kg}{m^3}$ at the bottom (1,64 m). The density profile show two points where a knee occur. The first knee at depth 0,46 m mark the spot where temperature start to drop at a steady rate, which give an equivalent increase in water density. The second knee mark a point near the bottom of the tank where water temperature has a slight increase that cause a sudden decrease in water density from 1.38 m to 1.64 m. The increase temperature is probably caused by the warm air surrounding the glass fiber tank.



(a)



(b)

Figure 8.6: Density and temperature measured with CTD instrument - **a:** Density **b:** Temperature.

Based on the density profile the Outer Lid position is set to 9,0 cm from the Vehicle House bottom, according to Table 6.1. The vehicle is found to be sinking until adjusted to 9.3 cm, approximately 3 mm more than calculated when measured with a ruler with 1 mm accuracy.

With the approximated $h_{add_{bot}} \approx 8.18$ cm for 998 $\frac{kg}{m^3}$, from Børseth's table 4.2[3, p. 36], equation 2.14 from Børseth's thesis[3, p. 13] is used to make a control calculation. The additional Outer Lid position is calculated to compensate for added weight, without taking volume into consideration:

$$h_{add} = \frac{m_{load_{fw}}}{\rho_f \pi r^2_{vehicle}}$$

Where m$_{load_{fw}}$ is the added load, ρ_f water density (and desired vehicle density to make it neutral buoyant and floating), and $r_{vehicle}$ is the vehicle radius.

$$h_{add} = \frac{0,141kg}{997.65\frac{kg}{m^3}\pi(0,065m)^2}$$

$$h_{add} = 1,06cm$$

Adding 8.18 cm with the additional height $h_{add} = 1,06cm$ give $h_{add_{bot}} \approx 9.24\ cm$.

The calculation without volume change is closer to the experienced Outer Lid position at neutral buoyancy. This indicate that the added weight is correct, while there may be a miscalculation related to the new total volume.

8.2.2 Functional Performance

This section present and discuss significant observations of the vehicles functional performance when tested in water.

Battery and Failure State

As battery is drained to 12,8 V the mission is aborted and LOW-POWER state is entered indicated by a bright red LED and vehicle float up to the surface to be picked up. Likewise whenever a system error occur the mission is aborted and failure-state is triggered. This is indicated by a pink LED, and the vehicle float to the surface.

8.2.3 Unknown Random System Errors

There are a number of cases where the system enter failure-state during assembly or adjustment of the outer lid, or turned up-side-down to let air out of the piston when diploid for mission. As errors are not logged it is not known what cause the errors triggering failure-state. The behavior is attempted replicated in debug mode to analyze and find the root cause, without success. From the behavior seem to appear in relation to rough handling of the vehicle. The system may be vulnerable to flickering on the power supply, caused by a loose battery connector or if batteries themselves are subject to movement. Other connectors or components may also cause flickering or disturbance when the vehicle is handled and movement is involved. This is however difficult to check without a clear indication and replication as to where or when the faulty behavior is triggered. An oscilloscope measuring different connection points while the vehicle is subject to movements may give an indication to problems related to flickering. Otherwise an error log that log any event that trigger the failure state have potential to give a clear, or at least a good indication of the software area that triggered the failure-state. As the failure-state always abort mission the error may be logged in the mission Log file. Otherwise, a dedicated error Log file can be generated upon reset that log any kind of error or abnormal events.

Float-To-Surface Failure

During test in water a functional error occur approximately every 5-6 test runs. When all missions are finished and vehicle is supposed to float to the surface, before state is changed to Pick-Up, the piston is not moved to zero-position (all the way out). The failure-state is not triggered, which indicate that this is a logical/code bug, and not a system induced bug. It appear to be a bug where either the motor is never starting to move the piston out, or the motor start to move the piston but is stopped immediately. The behavior is attempted replicated in dry conditions, while in debug mode, without success. The behavior only occur when tested in water. It's eventually found that float-to-surface code section is called each time a mission ended, instead of after all missions has ended. The PID is therefor still running and it may be that the PID calculate one or more values after the float-to-surface command is sent, and therefor counteract and effectively cancel the float-to-surface operation. Correcting the code improve the faulty behavior to occur only approximately once every 30 times. However this a highly problematic bug that introduce a severe risk of losing the vehicle if operated freely in deeper water. A watchdog timer need to be implemented that check for measured depth a defined maximal time after mission is finished, and if it is found to be still submerged the piston is force to go all the way to the bottom limit switch.

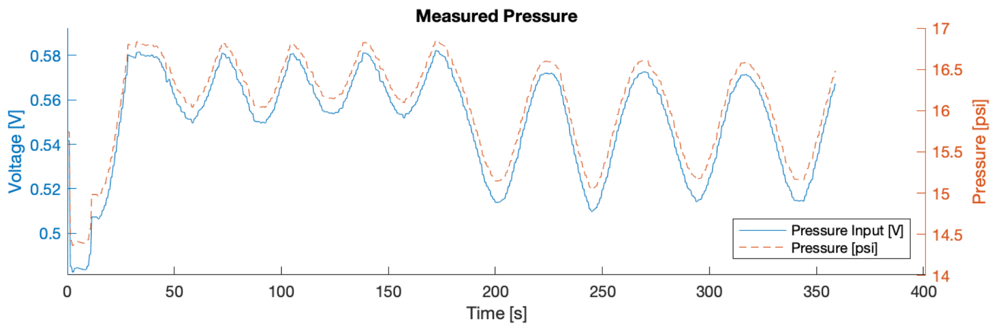
8.2.4 PID Tuning

Test Case 1:

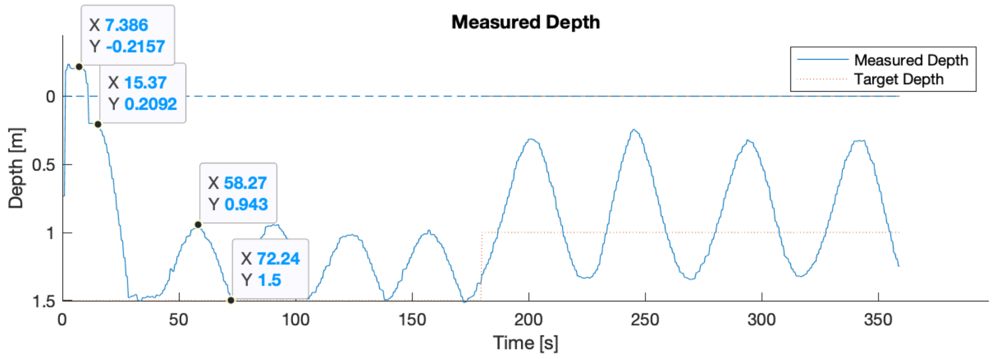
Figure 8.7a, b, and c show the pressure sensor input, the vehicle behavior with respect to depth, and piston position compared to PID output values. There are several interesting points to be made. The dive was started at the side of the tank which is 20-30 cm higher than the water surface. The measured depth therefor show ca -20 cm offset. Once in the water, which is shown as the knee at $X = 15.37$ seconds the depth is measured to 20,95 cm. As the pressure sensor sits at the bottom of the vehicle, approximately 39,3 cm (30 cm + 9,3 cm) beneath the measured depth has an offset of approximately 10 cm. This is also seen as the vehicle hit bottom at approximately 1,6 m (the bottom of the tank is not flat, so depth is varying with approximately 5 cm across the test area), while measured 1,5 m. This is due to the atmospheric pressure at the time test is carried out. The vehicle assume an atmospheric pressure of 1 atm = 14.7 psi, while the actual pressure is, according to the local weather station, 1026 hpa $\approx$ 14.88 psi. As the gauge pressure sensor experiences 14,88 psi, but software adjust input voltage for only 14,7 psi a slightly higher voltage is needed to represent the depth of 1,6 m. Measured depth is therefor calculated to 10 cm less than it actually is.

As seen in Figure 8.7b, the set point is perceived the bottom of the tank. The vehicle appear to attempt oscillating around set point (1,5 m), hitting the bottom (measured 1,5 m) before it float up 50 cm, then again hit the tank floor. The same oscillating pattern is seen when the program proceed to mission 2 with set point depth at 1.0 m. This time it has the free space to fully oscillate around the 1,0 m set point, varying between 0,4 m and 1,4 m (with a 10 cm error due to perceived measured pressure).

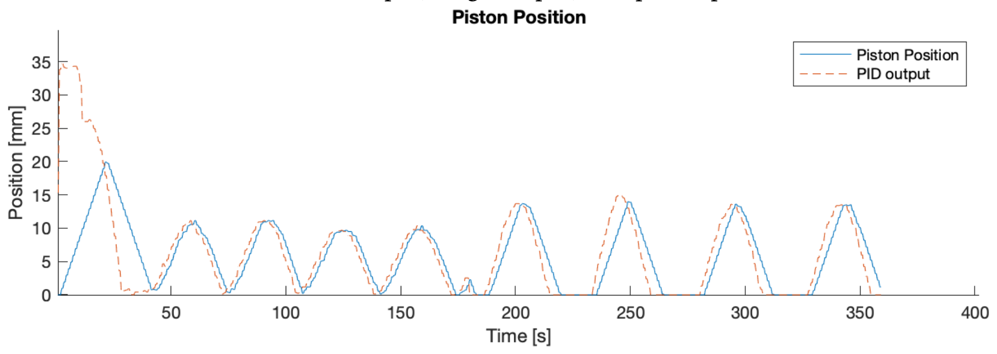
The piston position, in Figure 8.7c, have the same oscillating pattern with a 180°. The similar oscillating pattern indicate to rather than counteract it help to uphold it. The piston follow PID output with only slight delay and a rounded shape, which indicate that motor have room for a more aggressive PID tuning. On the second mission the piston position is more like a half saw-tooth pattern and does not follow PID output as close as before. This indicate that the motor has maxed out its ability to move the piston, and the vehicle fast enough to create a stabilizing effect in the system.



(a) Pressure input voltage and pressure converted to PSI



(b) Measured depth, target depth, and piston position



(c) Piston position and PID output value

Figure 8.7

Test Case 2:

As set point at 1,5 m cause the vehicle to hit the bottom and thereby create a dampening effect a new set point is placed at 1,3 meter and PID is adjusted by trial and error over a few tests. The atmospheric pressure is also adjusted to 15,02 psi as the given current atmospheric pressure, 14,88 psi, is found to be too little to measure correct depth. The vehicle is configured with the following values:

- P: 0,02
- I: 0,01
- D: 0,01
- Atmospheric Pressure: 15.02 psi

From Figure 8.8 it can be seen that The oscillation swing is reduced to set point, at 1,3 m, ± 0,3 m. The vehicle was observed to stop just above the bottom of the tank while it was measured to 1,7 the lowest, which indicate the pressure offset is too high.

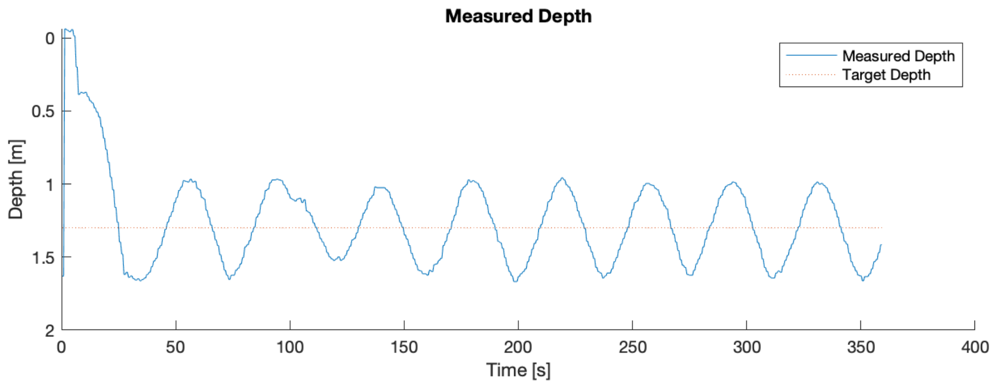


Figure 8.8: Measured depth for set point at 1,3 m

Test Case 3:

The last key test has set point at 1,0 m, a slight decreased atmospheric pressure at 14,99 psi, and the following PID coefficients:

- P: 0,02
- I: 0,01
- D: 0,05

Figure 8.9 show oscillation with a slight dampening. To increase the dampening effect the derivative D need to be increased even more.

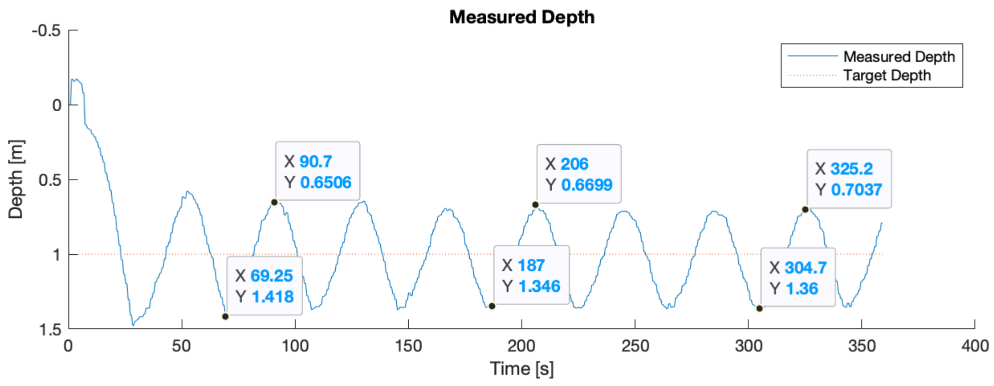


Figure 8.9: Measured depth for set point at 1,0 m

8.3 Fresh Water Lake

This section present and discuss the results from test in freshwater lake. The obtained PID parameters and system response is presented and discussed, followed by a presentation of the measured battery voltage; where a rough estimation of expected battery life during mission is calculated.

8.3.1 Tuned PID

The following PID tuning parameters are found to give the optimal system response, shown in Figure 8.10:

- P: 0.04
- I: 0.01
- D: 0.07

The lake is 184 meter above sea level. To measure correct depth the atmospheric pressure is adjusted accordingly to 14.6 psi, slightly below 1 atm.

Figure 8.10 show the measured depth (blue line) and target depth (red dotted line), PID set point, of the dive with optimal PID tuning. The dive has two missions: Mission 1 with target depth 1,5 m for 180 sec, mission 2 with traget depth 1,0 m for 180 sec.
There is a 5 second negative top initially caused by the mission being started while on land, and the vehicle is lifted and carried to the water before it descend towards target depth. When target depth is reached It overshoot with 25 cm before it regulate to set point. Steady state is reached after a settling time of 95 sec as a consequence of an inherently slow system due to its design. To create a faster system response a bigger piston volume is needed, which create greater difference in vehicle density to regulate the effect of buoyancy force acting on the vehicle.

A new target depth is set with the starting of mission 2 at 180 Sec, and the vehicle is floating up to 1,0 m with an overshoot of approximately 20 cm before it regulate towards set point. The settling time is shorter, approximately 75 sec, as the vertical distance is $\frac{1}{3}$ of that of mission 1. The vertical position is well regulated until mission 2 is finished.

Figure 8.11 show the PID output value and the resulting piston position set accordingly. During the first 15 seconds the PID output reach its upper limit at 0,055 and the piston position is moved up until it reach the limit at $\approx$ 51,6 mm. the vehicle is regulated to hold target depth by adjusting piston position up and down by 5 - 10 mm centered between 40 mm and 35 mm. When new set point is given at time 180 s the response is shown as a PID output spike to initiate the ascending towards the new set point. Piston position reach the approximate same position after a slight delay, though by the time piston position has moved, the PID has calculated a slightly higher value so the piston position spike is not as low as the PID output. The low spike is followed by an equally high spike to counteract the overshoot. At time 250 s the 5 - 10 mm regulating behavior is seen, that hold the vehicle at set point.

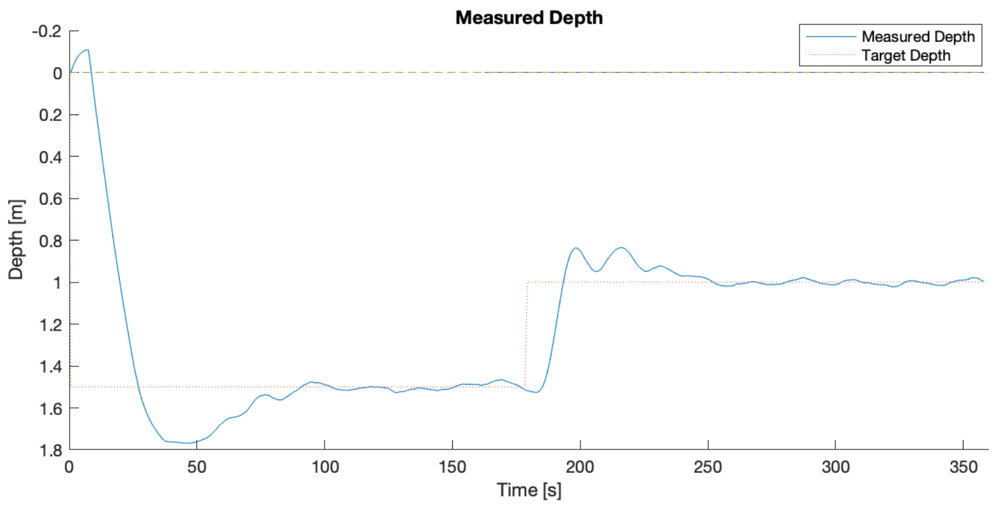


Figure 8.10: Measured Depth with optimal PID tuning parameters

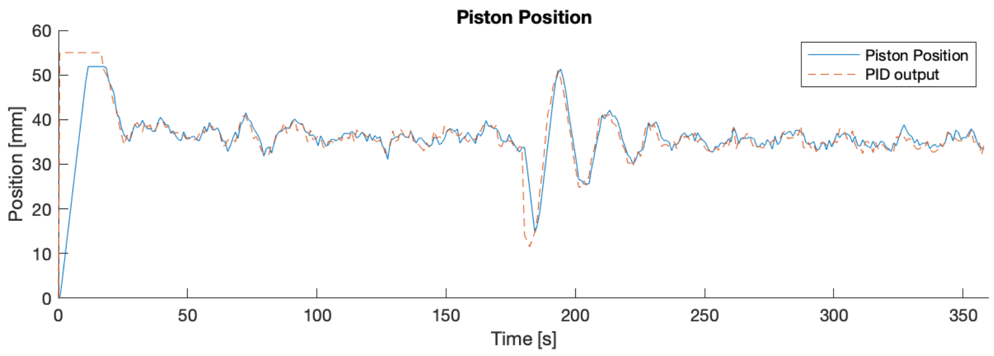


Figure 8.11: PID output and piston position

8.3.2 Battery Power

Figure 8.12 show the measured battery voltage during the mission discussed in subsection 8.3.1. Each time motor is adjust the piston position there is a voltage drop of ≈ 1 V. Assuming linear voltage drop as the batteries are drained, the graph can provide a rough estimation of how long the vehicle can operate on a mission, by measuring the minimum voltage at the beginning and end of the dive. The initial 1,5 v drop is not part of the calculation as it is not a result of the dive it self, but rather to show the effect of added torque, but applying pressure to the piston before it was put in the water. At the beginning of the mission a low value is measured to 18,98 V, the two following measurements are highlighted to show how the voltage is falling steadily. At the end of Mission 2 the battery voltage is 18,82 V. The average voltage drop per minute can then be calculated:

$$V_{dropPerMin} = \frac{18,98 - 18,82V}{6min} \approx 26,67\frac{mV}{min}$$

and the estimated operation time by dividing the total voltage span (subtracting 1 V from high end to account for motor voltage drop):

$$t_{operation} = \frac{23,0 - 12,8V}{26,67\frac{mV}{min}} \approx 382,45min \approx 6,37hrs$$

As this is a rough calculation based on a 6 min dive further testing need to be done. Example a longer dive of 30 - 60 minutes to better see the effect of battery voltage drop.

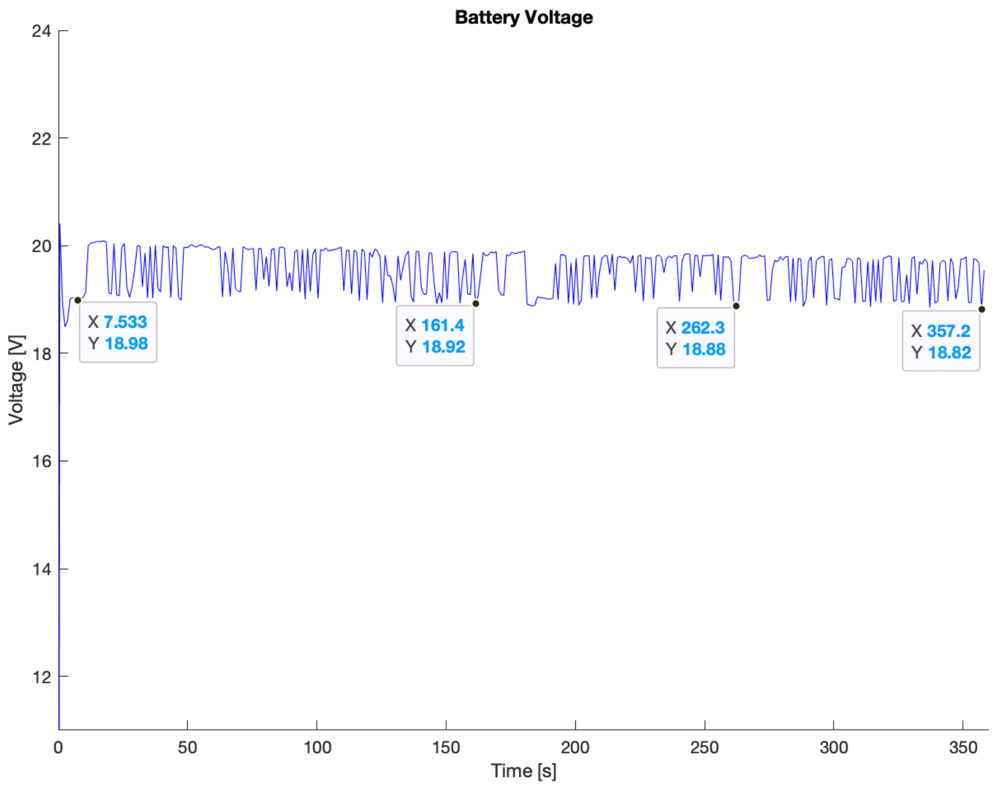


Figure 8.12: Measured battery voltage level through mission

8.4 Salt Water

Figure 8.13a and b show the results from a test dive with two missions in salt water.

8.4.1 Mission 1

From Figure 8.13a the vehicle oscillate around set point at 1.0 m with increasing amplitude, which indicate instability. During the same time frame in Figure 8.13b the same behavior is seen, where the PID controller calculate increasingly higher values than the motor is able to reach before a new value is calculated in the opposite direction. This cause the motor to move the piston head up and down in a saw-tooth like pattern, which result in the vehicles oscillating behavior. The exact reason for the behavior is not known, though the PID output seem to be way more aggressive than the motor is able to handle. If this is the case it may seem to be a problem with the system itself, and not the fact that it is tested in salt water.

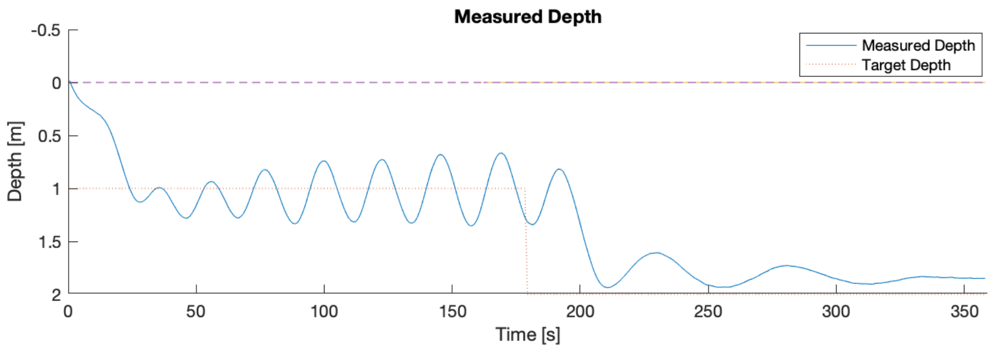
Salt water is more dense than fresh water, a greater buoyancy force is therefore acting on the vehicle, according to Equation 2.1, however the outer lid is adjusted for the added density. As the vertical force balance is a result of the difference between water density and vehicle density, expressed in Equation 2.2, the higher density in salt water will cause the density difference to increase accordingly, which result in a greater force (up or down) as the vehicle is regulated. The increased force may cause the vehicle to move faster in the water, which is picked up by the PID controller and cause the derivative D term to calculate extreme values. If this is the case the PID need different set of tune parameters for salt water. Or the PID controller need to be more robust to be able to handle the inconsistencies.

The depth/set point is configured different than in the fresh water lake test. It is a possibility that either this fact alone, or the combined effect of the properties of salt water and the greater difference between set points is the causing factor of the unstable behavior. Another test in salt water should be made with mission 1 depth at 1,3 m, and mission 2 depth at 1 m, where results are compared to this test, and the fresh water lake test.

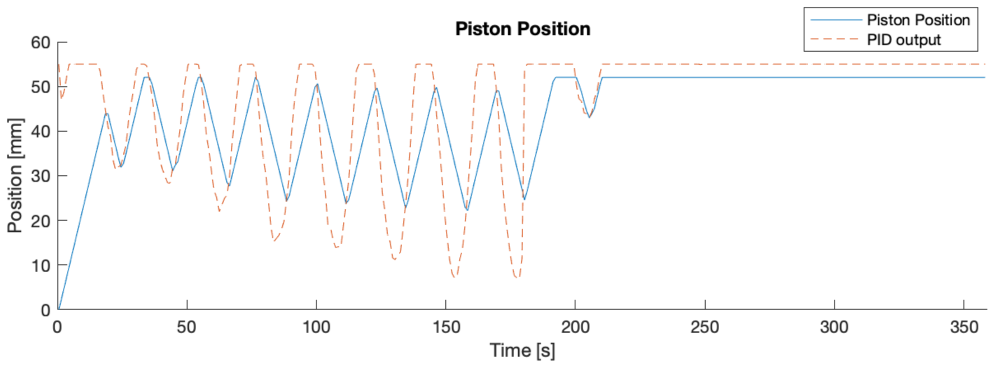
The same PID parameters should also be tested again both in fresh water and salt water to reassure the parameters are correct, or if other factors may have caused a disturbance.

8.4.2 Mission 2

At the next set point, 2 m, another interesting phenomenon is observed. The vehicle appear to settle quite well 10 cm above 2 m. However, the piston position reveal that the actual reason is the vehicle is not able to dive any deeper, as the water density at measured 1.9 meter is equal to the vehicles maximal density. A density profile would be a benefit, to know before hand the required maximal density of the vehicle, and to afterwards be able to analyze the vehicle response.



(a) Target depth and measured depth



(b) PID output and piston position

Figure 8.13

8.5 Wireless Connection In Water

8.5.1 Test 1: Signal strength On land

Figure 8.14 show results from signal strength on land in a fresh water environment versus salt water environment, and with chip antenna versus FlexPIFA$^{\text{TM}}$ antenna. The test show a clearly improved signal strength with FlexPIFA$^{\text{TM}}$ antenna over chip antenna. It does not give a clear indication as to what effect the surrounding area has on the signal strength. The saltwater curves have a dip between 0,5 and 2 meters, though more characteristic for chip antenna, that is unclear as to what cause it. A possible explanation could be interference from nearby frequencies, or reflections. While the chip antenna range seem to be near the point of no connection at 16 meters the FlexPIFA$^{\text{TM}}$ antenna still have a ways to go before it hit the -90 to -100 dBm area.

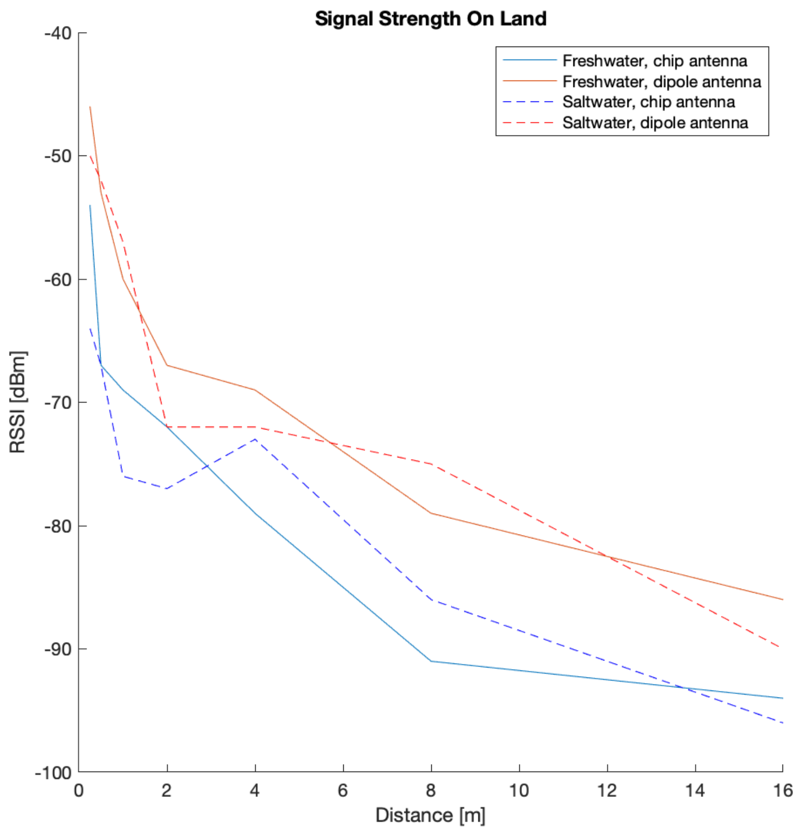


Figure 8.14: Test results to show RSSI as the vehicle is on land at fresh water test location and salt water test location. The test is performed with chip antenna and FlexPIFA™ antenna

8.5.2 Test 2: Signal Strength When Floating At The Surface

Figure 8.15 show results from signal strength when the vehicle is floating in the surface of fresh water and salt water, and tested with chip antenna and FlexPIFA™ antenna. The FlexPIFA™ antenna provide a far better signal compared to chip antenna. And by large the results from salt water prove better than that of fresh water. This stand in contradiction to theory that says that salt water should cause more resistance, caused by reflection of the propagated waves, than fresh water. A possible explanation here is that the vehicle is floating a bit higher in the salt water experiment than the fresh water experiment. If this is the case there is less water to propagate through, which give better signal strength even with salt water.

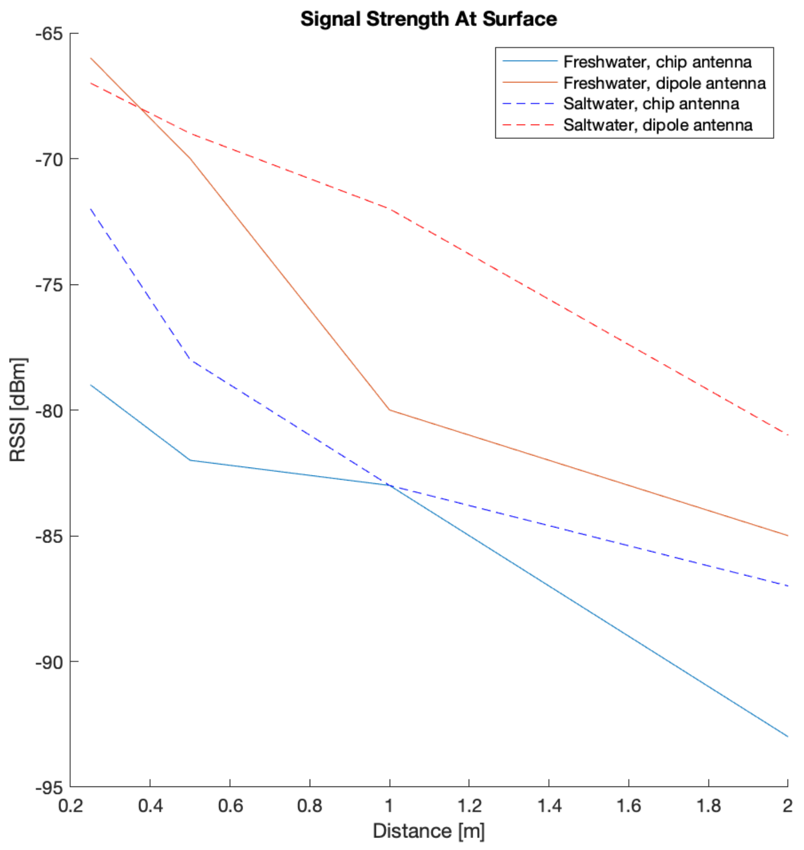


Figure 8.15: Test results to show RSSI as the vehicle float in the surface. The test is performed in fresh water and salt water, and with chip antenna and FlexPIFA™ antenna

8.5.3 Test 3: Signal Strength When Submerged In Water

Figure 8.16 show results from signal strength when vehicle is submerged in fresh water and salt water, fitted with chip antenna and FlexPIFA™ antenna. The curves end at the given maximal depth for which a signal was possible to measure. The FlexPIFA™ antenna perform better than chip antenna. Where maximal depth, in centimeter, is 14 cm and 5 cm for fresh water and salt water respectively. Maximal depth for chip antenna is 13 cm and 3 cm for fresh water and salt water respectively. As expected from theory fresh water is clearly better in terms of signal strength than salt water. The first 2 centimeters yield similar signal strength, however as depth increases the difference is dramatic. as the vehicle can be submerged to 13 - 14 cm in fresh water and still measure the signal, the vehicle reach only 3 - 5 cm in salt water.

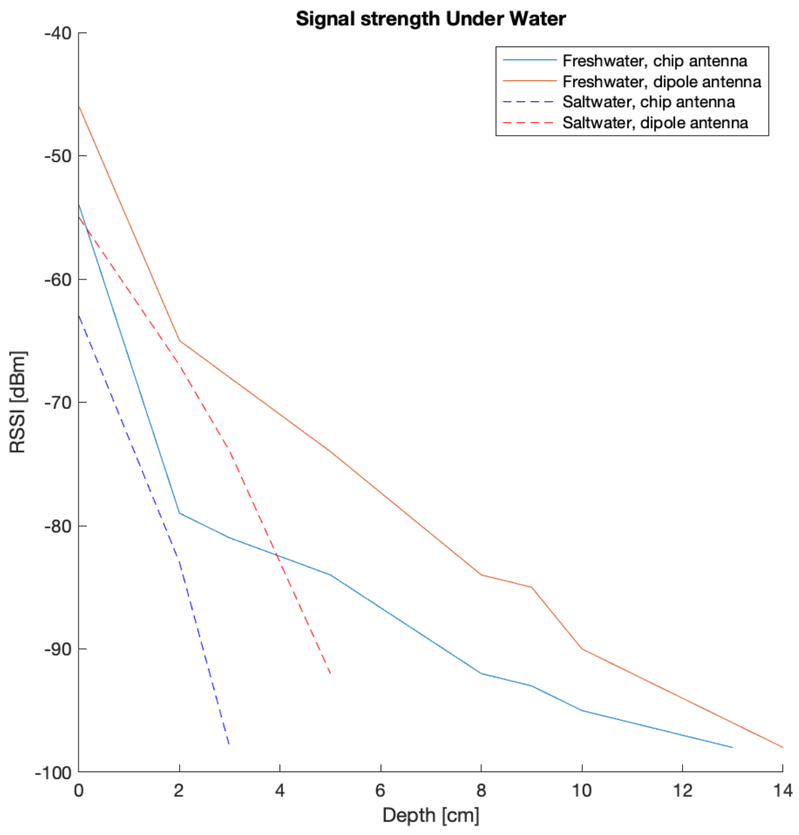


Figure 8.16: Test results to show RSSI as the vehicle is submerged in water. The test is performed in fresh water and salt water, and with chip antenna and FlexPIFA™ antenna

8.5.4 Test 4: Data Transmission Rate When Submerged In Water

Figure 8.17 show the results from data rate measurements, where a data file is transferred while vehicle is submerged under water. Fresh water yield far better results than do salt water. With FlexPIFA™ antenna the data rate remain at around 4000 B/sec until depth at 9 cm in fresh water. The last successful data transmission happens at 10 cm below surface at around 2500 B/Sec. The chip antenna have a steady decline from 3 cm to 8 cm down to ca 1500 B/Sec where it suddenly, for unknown reason, increase to ca 3000 B/Sec. before on the next measurement connection is lost. In salt water the maximal depth for a successful data transmission is 5 cm with FlexPIFA™ antenna, and 3 cm with chip antenna. These are still promising results as the vehicle will be floating at around 2 cm under the surface. When chip antenna is used only small disturbances at the surface will case the trouble for data transmission. With a FlexPIFA™ antenna, however, there is room for even small waves and still be able to transfer a file.

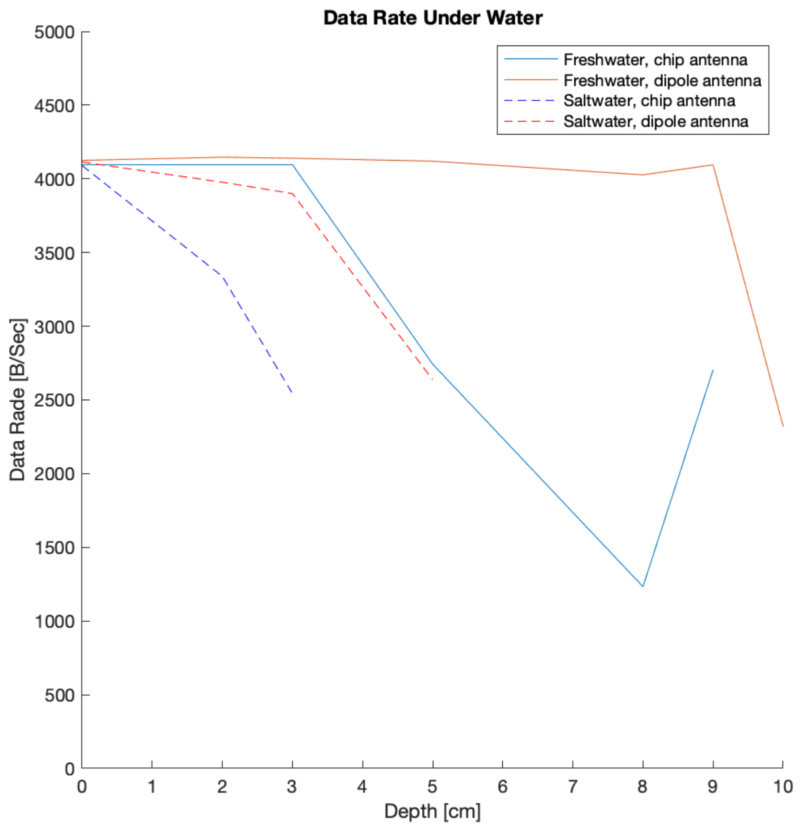


Figure 8.17: Test results for measured data rate of transferred data file from vehicle submerged in water

8.5.5 Test 5: Signal Strength At Oblique Angle

Table 8.3: Test results to show the effect on wireless connection from an angle when vehicle is floating in the surface

Chip antenna:	Fresh Water	Salt Water
Max Distance:	2,5 m mA	3,4 m
Angle:	36,86°	28,07°
Data rate:	4096 B/Sec	1519 B/Sec
FlexPIFA™ antenna:		
Max Distance:	3,8 m	4,5 m
Angle:	23,19°	26,56°
Data rate:	1976 B/Sec	2215 B/Sec

Table 8.3 show the results in terms of maximal distance at an angle where connection is obtained, the angle with respect to the water surface, and the data rate

at a successful file transmission. FlexPIFA™ antenna perform better than chip antenna. It manages to transfer a file at longer distance and smaller angle than the chip antenna. It is difficult to tell what difference the salt water and fresh water has in this regard. The angles are not exactly the same, nor are the distance, or data rates. It seem that the performance is better in salt water, as the distance is longer and the angle is smaller. At least for chip antenna. A better test would be to hold either the angle stationary to see the effects on data rate and maximal distance.

8.6 Future Work

The testing of the implemented sensor- and control system have revealed promising results, some challenges that need to be addressed, and some new opportunities for further development:

- The most important of these are how to make the vehicle reliable enough to be deployed on missions without a safety line attached. The piston have to reach minimum density after a dive every time. As stated in section 8.2.3 a watchdog timer need to be implemented to bring the vehicle to surface, in case it remain submerged after a certain time.

- The PID controller is not yet optimally tuned. Parameters with promising response is found for freshwater, though when tested in salt water the response indicate instability. Further test and tuning need to be done both in fresh water and salt water. There is also room for increased piston speed, though motor current probably need to be increased as well to ensure enough torque to handle 50+ meters, as discussed in subsection 5.10.3.

- The transmission data rate at 4 kB/sec have a lot of room for improvement. The code should be reviewed and optimized for file transmission data rate. This will be important when missions become longer and more data is logged.

- TMP117 and ICM20948 sensors are not yet configured. These make up an important part of the data acquisition capabilities of the vehicle.

- As test results show reliable wireless connection at depth down to 5 cm in salt water, a study should be made to look in to the possibility of wireless data transmission between an external sensor, attached under the vehicle, and the vehicle itself.

- As per now the vehicle is not fully optimized for low power operation. A lot can be done to limit power consumption. Most of all, the motor interface need to be powered down, when inactive. Then, the nRF52832 SoC can be optimized for low power during sleep state and low power state.

- A method to reduce the need for calibration of the atmospheric pressure compensation to measure correct depth, such as self calibration, will make the system more robust and faster to setup for mission.

Chapter 9

Conclusion

The obtained results testify to the successful implementation of the embedded sensor- and control system in a buoyancy vehicle designed for under water data acquisition. The existing solution is reviewed in a use case analysis, which resulted in a list of system specifications, found in Appendix B. The finished design satisfy the specifications where the functional test verify the working of core operation functionality, such as system state indicated by LED; system state changed by a magnetic activated button; and a Bluetooth interface to transfer data files, configure vehicle, and change system state. Various mechanical modifications have improved the safety and user friendliness of the vehicle, such as end limit switches, indication LED, and a protective cover for the linear actuator. SoC utilization is on-point for a prototype project with GPIO count, FLASH, and RAM utilization at 50-65 %, which leave available resources for further development. A measurement of current consumption reveal that, apart from active use of the motor, the motor controller interface is the main source of power consumption, with a current consumption of 55 mA while motor is inactive. Test in fresh water reveal promising results as the PID controller is tuned to maintain target depth; However, the tuning obtained in fresh water behave differently in salt water; where the system response is characterized by increasing oscillation, centered around target depth, which indicate unstable behavior – not suitable for vehicle operation. The vehicle fail to float back to surface once every 30 mission deployment. The causing factor remain unknown and need further investigation as it introduces a risk of losing the vehicle under water. The wireless properties of air-to-water interfacing are investigated for both fresh water and salt water, as well as a comparison of two antenna configurations. It is found that a FlexPIFA give the better signal strength, over a ceramic monopole chip antenna; where reliable connection is maintained at depths down to 10 cm in fresh water, and 5 cm in salt water. This is promising as the vehicle can be accessed for file transmission and configuration while floating in the water surface. however, there is a risk of unreliable connection in disturbed water – especially in salt water. The vehicle was originally planned to be tested in Aquatrass, a large scale sea cage, to demonstrate the ability to track current flow; however this was not an available option at the time of testing. The main goals have been met, and is realized as a system that fulfill the system specifications.

Possible future developments of the vehicle and on-board system consists of ensuring the enabling of a fully independent mission deployment, where the vehicle is verified floating to the surface every time a mission deployment has ended; further testing in fresh- and salt water to find PID parameters with optimal system response; test the limitations of wireless connection in fresh- and salt water in rough conditions, such as waves; and optimize the system for low power consumption to extend the battery life during operations, and to maximize storage life without the need to disconnect battery power.

Bibliography

[1] J. S. Jaffe, P. J. Franks, P. L. Roberts, D. Mirza, C. Schurgers, R. Kastner and A. Boch, 'A swarm of autonomous miniature underwater robot drifters for exploring submesoscale ocean dynamics', *Nature Communications*, vol. 8, pp. 1–8, 2017, ISSN: 20411723. DOI: 10.1038/ncomms14189.

[2] A. Joseph, 'Lagrangian-Style Surface Current Measurements Through Tracking of Surface Drifters', in *Measuring Ocean Currents*, Elsevier, 2014, pp. 93–107. DOI: 10.1016/b978-0-12-415990-7.00003-x.

[3] S. Børseth, 'Low Energy Buoyancy Actuator for Vertical Underwater Motion', PhD thesis, Norwegian University of Science and Technology, 2017. [Online]. Available: https://pdfs.semanticscholar.org/0e98/ab062c1ff5a3c282fe86467cb74f54393812.pdf?_ga.

[4] R. A. Serway and J. W. Jewett, *Physics for Scientists and Engineers with Modern Physics*. Cengage learning, Apr. 2012, ISBN: 978-1-133-95405-7. [Online]. Available: https://books.google.no/books?id=wVqBa_AR6nkC&pg=PA424&dq=buoyant+force&hl=en&sa=X&ved=0ahUKEwi9jLyHlv7nAhUzAxAIHfaUBAoQ6AEIQjAD#v=onepage&q=buoyant%20force&f=false.

[5] B. Lautrup, *Physics of Continuous Matter: Exotic and Everyday Phenomena in the Macroscopic World*. Copenhagen: Institute of Physics Publishing, Bristol and Philadelphia, 2005, ISBN: 0-7503-0752-8. [Online]. Available: https://books.google.no/books/about/Physics_of_Continuous_Matter.html?id=s1265-uY-BgC&printsec=frontcover&source=kp_read_button&redir_esc=y#v=onepage&q&f=false.

[6] R. L. Saunders, 'Adjustment of Buoyancy in Young Atlantic Salmon and Brook Trout by Changes in Swimbladder Volume', *Journal of the Fisheries Research Board of Canada*, vol. 22, no. 2, pp. 335–352, Feb. 1965, ISSN: 0015-296X. DOI: 10.1139/f65-034.

[7] W. J. Finney and R. L. Brown, 'By 242-ee ATTORNEY', United States Patent Office, Tech. Rep., 1961. [Online]. Available: https://patentimages.storage.googleapis.com/dd/b4/f2/a563752d8efd63/US2968053.pdf.

[8] D. C. Sparks, L. Belfie, D. Bruder, C. T. Werner and J. W. Widenhofer, 'Sparks et al 2012', 1993. [Online]. Available: https://patentimages.storage.googleapis.com/15/b6/14/217c246c74befe/US5379267.pdf.

[9] R. E. Davis, L. A. Regier, J. Dufour and D. C. Webb, 'The Autonomous Lagrangian Circulation Explorer (ALACE)', *Journal of Atmospheric and Oceanic Technology*, vol. 9, no. 3, pp. 264–285, Jun. 1992, ISSN: 0739-0572. DOI: 10.1175/1520-0426(1992)009<0264:TALCE>2.0.CO;2.

[10] H. Yamamoto and K. Shibuya, *New small buoyancy control device with silicone rubber for underwater vehicles*, Sep. 2016. DOI: 10.1299/jsmermd.2016.1a2-17a3. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/9789813149137_0032.

[11] M. R. Clarke, 'Buoyancy control as a function of the spermaceti organ in the sperm whale', *Journal of the Marine Biological Association of the United Kingdom*, vol. 58, no. 1, pp. 27–71, 1978, ISSN: 14697769. DOI: 10.1017/S0025315400024395. [Online]. Available: https://core.ac.uk/download/pdf/78757343.pdf.

[12] M. Gudino, *What is a Microcontroller? A Look Inside a Microcontroller | Arrow.com | Arrow.com*. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/engineering-basics-what-is-a-microcontroller (visited on 03/02/2020).

[13] *Embedded Systems - Overview*. [Online]. Available: https://www.tutorialspoint.com/embedded_systems/es_overview.htm (visited on 09/07/2020).

[14] *PID Theory Explained - National Instruments*, 2011. [Online]. Available: https://www.ni.com/en-no/innovations/white-papers/06/pid-theory-explained.html (visited on 08/06/2020).

[15] C. Foster, *Control Engineering | Modern updates in PID control tuning*. [Online]. Available: https://www.controleng.com/articles/modern-updates-in-pid-control-tuning/ (visited on 27/06/2020).

[16] J. S. Cook, *PID Controller Basics & Tutorial: PID Arduino Project | Arrow.com*. [Online]. Available: https://www.arrow.com/en/research-and-events/articles/pid-controller-basics-and-tutorial-pid-implementation-in-arduino (visited on 09/06/2020).

[17] J. Lloret, S. Sendra, M. Ardid and J. J. P. C. Rodrigues, 'Underwater Wireless Sensor Communications in the 2.4 GHz ISM Frequency Band', *Sensors*, vol. 12, no. 4, pp. 4237–4264, Mar. 2012, ISSN: 1424-8220. DOI: 10.3390/s120404237.

[18] U. Qureshi, F. Shaikh, Z. Aziz, S. Shah, A. Sheikh, E. Felemban and S. Qaisar, 'RF Path and Absorption Loss Estimation for Underwater Wireless Sensor Networks in Different Water Environments', *Sensors*, vol. 16, no. 6, p. 890, Jun. 2016, ISSN: 1424-8220. DOI: 10.3390/s16060890.

[19] S. Jiang and S. Georgakopoulos, 'Electromagnetic Wave Propagation into Fresh Water', *Journal of Electromagnetic Analysis and Applications*, vol. 03, no. 07, pp. 261–266, 2011, ISSN: 1942-0730. DOI: 10.4236/jemaa.2011.37042.

[20] *nRF52832 Product Specification*. [Online]. Available: https://infocenter.nordicsemi.com/index.jsp?topic=%2Fstruct_nrf52%2Fstruct%2Fnrf52832_ps.html.

[21] J. Kaye and R. Khatri, 'Laird BL652 Datasheet', Tech. Rep., 2016. [Online]. Available: http://assets.lairdtech.com/home/brandworld/files/Datasheet%20-%20BL652.pdf.

[22] Advanced Ceramic X Corp., 'AT3216 Series Multilayer Chip Antenna datasheet', Tech. Rep. [Online]. Available: https://www.acxc.com.tw/_upload/files/AT3216-A2R8HAA_S-R00-N200_6.pdf.

[23] *2.4 GHz FlexPIFA Antenna, 100mm Datasheet*, 2017. [Online]. Available: https://connectivity-staging.s3.us-east-2.amazonaws.com/s3fs-public/2018-10/Datasheet%20-%202.4%20GHz%20FlexPIFA.pdf.

[24] S. Labs, 'Si7210 I 2 C Hall Effect Magnetic Position and Temperature Sensor Data Sheet', Tech. Rep. [Online]. Available: https://www.silabs.com/documents/public/data-sheets/si7210-datasheet.pdf.

[25] 'TMP117 datasheet', Tech. Rep. [Online]. Available: https://www.ti.com/lit/ds/symlink/tmp117.pdf.

[26] *High Precision Temperature Sensor - TMP117 (Qwiic) - SPX-15413 - SparkFun Electronics*, CC BY 2.0: https://creativecommons.org/licenses/by/2.0/. [Online]. Available: https://www.sparkfun.com/products/retired/15413 (visited on 10/07/2020).

[27] 'ICM-20948 datasheet', Tech. Rep. [Online]. Available: https://invensense.tdk.com/wp-content/uploads/2016/06/DS-000189-ICM-20948-v1.3.pdf.

[28] 'Heavy Duty Pressure Transducers, PX3 Series, 1 bar to 50 bar | 15 psi to 700 psi', Fort Mill, Tech. Rep., 2019. [Online]. Available: https://sensing.honeywell.com/honeywell-sensing-heavy-duty-pressure-px3-series-datasheet-32313757.pdf.

[29] '3W RGB LED Module', Tech. Rep. [Online]. Available: https://cdn.sparkfun.com/assets/e/c/9/5/d/M018001MA3LZ.pdf.

[30] E. Zonca, 'PicoBuck Hookup Guide V12', Sparkfun, Tech. Rep. [Online]. Available: https://media.digikey.com/pdf/Data%20Sheets/Sparkfun%20PDFs/PicoBuck_HookupGuide_V12_Web.pdf.

[31] *PD-1161 V1.0 Hardware Manual*, Jul. 2013. [Online]. Available: https://www.trinamic.com/fileadmin/assets/Products/Drives_Documents/PD-1161_hardware_manual.pdf.

[32] 'MTCL 1161 Firmware Manual', Hamburg, Tech. Rep., 2018. [Online]. Available: https://www.trinamic.com/fileadmin/assets/Products/Modules_Documents/TMCM-1161_TMCL_firmware_manual_Fw1.42_Rev1.08.pdf.

[33] 'D2MQ Subminiature Basic Switch', Tech. Rep. [Online]. Available: https://omronfs.omron.com/en_US/ecb/products/pdf/en-d2mq.pdf.

[34] *MCP602 Datasheet*. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/21314g.pdf.

[35] *ENERGIZER EN93 Product Datasheet*. [Online]. Available: https://data.energizer.com/PDFs/EN93.pdf (visited on 09/07/2020).

[36] *Traco TSR 1-2450 Datasheet*. [Online]. Available: https://www.tracopower.com/products/tsr1.pdf.

[37] *LM1117 800-mA, Low-Dropout Linear Regulator Datasheet*. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm1117.pdf.

[38] *nRF5 SDK downloads - nordicsemi.com*. [Online]. Available: https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK/Download (visited on 22/06/2020).

[39] *The Difference Between Classic Bluetooth and Bluetooth Low Energy*. [Online]. Available: https://blog.nordicsemi.com/getconnected/the-difference-between-classic-bluetooth-and-bluetooth-low-energy (visited on 27/06/2020).

[40] 'Introduction to Bluetooth Low Energy', [Online]. Available: https://embeddedcentric.com/introduction-to-bluetooth-low-energy-bluetooth-5/.

[41] A. R. D. Carles Cufí , Kevin Townsend, *1. Introduction - Getting Started with Bluetooth Low Energy*, 2014. [Online]. Available: https://www.oreilly.com/library/view/getting-started-with/9781491900550/ch01.html (visited on 27/06/2020).

[42] O. Jones, 'DESIGN AND DEVELOPMENT OF AN EMBEDDED DC MOTOR CONTROLLER USING A PID ALGORITHM', PhD thesis, Tekniska Högskolan Linköpings Universitet, 2015. [Online]. Available: http://liu.diva-portal.org/smash/get/diva2:347417/FULLTEXT01.pdf.

[43] *nRF52 SoftDevice description*. [Online]. Available: https://infocenter.nordicsemi.com/topic/struct_nrf52/struct/nrf52_softdevices.html?cp=4_6.

[44] J. Schneider, *BleTerm2*, 2019. [Online]. Available: https://github.com/josschne/bleterm2/.

[45] Timeular, *Noble-mac*, 2019. [Online]. Available: https://github.com/Timeular/noble-mac.

[46] M. Sandeep, *Noble: A Node.js BLE (Bluetooth Low Energy) central module*, 2015. [Online]. Available: https://github.com/noble/noble/ (visited on 27/06/2020).

[47] *nRF UART 2.0 - Apps on Google Play*. [Online]. Available: https://play.google.com/store/apps/details?id=com.nordicsemi.nrfUARTv2&hl=en (visited on 09/07/2020).

[48] K. Morich, *Serial Bluetooth Terminal - Apps on Google Play*. [Online]. Available: `https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=en` (visited on 09/07/2020).

[49] 'High efficiency 36v 1a buck led driver description', Tech. Rep. [Online]. Available: `https://www.diodes.com/assets/Datasheets/AL8805.pdf`.

[50] *SDK Drivers description: TWI Master*. [Online]. Available: `https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/hardware_driver_twi.html`.

[51] *SDK common libraries: UART module*. [Online]. Available: `https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/group__app__uart.html`.

[52] *SDK Hardware peripheral example: SD card*, 2020. [Online]. Available: `https://infocenter.nordicsemi.com/index.jsp?topic=%2Fsdk_nrf5_v16.0.0%2Fapp_sdcard_example.html`.

[53] *FatFs - Generic FAT Filesystem Module*, 2019. [Online]. Available: `http://elm-chan.org/fsw/ff/00index_e.html`.

[54] *SDK Drivers descriptions - SAADC*. [Online]. Available: `https://infocenter.nordicsemi.com/topic/sdk_nrf5_v16.0.0/hardware_driver_saadc.html`.

[55] Rubén Santa Anna, *PID controller algorithm C*, 2016. [Online]. Available: `https://github.com/geekfactory/PID/`.

[56] B. Beauregard, *Improving the Beginner's PID – Introduction*. [Online]. Available: `http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/`.

[57] *SDk common libraries: Application Timer*. [Online]. Available: `https://infocenter.nordicsemi.com/index.jsp?topic=%5C%2Fcom.nordic.infocenter.sdk5.v15.0.0%5C%2Fgroup__app__timer.html`.

[58] *Nrf52832 Product Specification: RTC - Real-time counter*. [Online]. Available: `https://infocenter.nordicsemi.com/index.jsp?topic=%5C%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%5C%2Frtc.html`.

[59] *App timer maximum time in SDK 15.2*. [Online]. Available: `https://devzone.nordicsemi.com/f/nordic-q-a/41532/app-timer-maximum-time-in-sdk-15-2`.

[60] M. Aasvik, *Tutorial: Potentiometers with Arduino and Filtering*, 2015. [Online]. Available: `https://www.norwegiancreations.com/2015/10/tutorial-potentiometers-with-arduino-and-filtering/` (visited on 28/06/2020).

[61] P. Kastnes, 'Power Consumption', in *Power Analysis Attacks*, Boston, MA: Nordic Semiconductor Company, 2017, pp. 27–60. [Online]. Available: `https://blog.nordicsemi.com/getconnected/power-consumption-explained`.

[62] *2N3904SC Datasheet*. [Online]. Available: `https://alltransistors.com/pdfdatasheet_upd/2n3904sc.pdf`.

Appendix A

Use Case

This appendix include the use cases defined in the use case analysis where the buoyancy vehicle is analyzed in terms of user expectations and needs. Vehicles possibilities, limitations, and various risk factors are also analyzed.

Bouyancy Vehicle Use Case

Case: Power-On

1. Brief Description

Starts when: Mode button is pressed

Ends when: Vehicle is initiated and in standby mode

2. Actors

- User
- MCU/system

3. Pre-Conditions

- The system is powered off
- Not in water

4. Basic Flow

1. Mode button is pressed
2. MCU/system is powered by batteries
3. Initialization code is executed
4. Standby mode is selected by default
5. Standby mode is indicated by status LED

5. Alternate/Exception Flows

Number alternate and exception flows to indicate the step at which the variation/exception occurs. For example, a variation on step 3 is listed as 3a and a second variation as 3b, and so forth.

2a – Battery is empty and MCU/system remain off

2b – Battery power is below a defined voltage level

3b – At some point during initialization the voltage is sampled, calculated to be below a set threshold, and the status LED is signaling low power

4b – The system is powered off

6. Post Conditions

- The system is powered on and in standby mode (indicated by status LED)
- The necessary drivers are initiated
- Exception: The status LED is signaling low power before system is powered off.

CASE: CONFIGURATION

1. BRIEF DESCRIPTION

Start when: Mode button is pressed and Configure-mode is selected.

Ends when: Mode is changed to either deploy-mode or standby-mode.

2. ACTORS

- User
- MCU/system
- Wireless protocol
- Computer with configuration interface
- Storage unit/Memory

3. PRE-CONDITIONS

- The system is initiated
- The system is in standby mode
- Sufficient battery power

4. BASIC FLOW

6. Mode button is pressed and configure-mode is selected
7. Config-mode is indicated by status LED
8. Wireless broadcast to connect to comptuer with configuration interface
9. Broadcast is indicated by rapidly blinking status LED
10. Connection to configuration interface is indicated by slow flashing LED
11. Configuration data is sent from BV to configuration interface
12. New configuration settings are typed in by user and sent to BV
13. BV receive new configuration setting
14. BV indicate received configuration settings by rapidly flashing status LED a few times, and sends an acknowledge/confirmation message back to computer user interface
15. BV stores new configuration setting
16. Wireless is powered off
17. either the mode button is pressed to select standby or deploy mode
    or the new configuration settings change to the new mode.

18. New mode is indicated by status LED

5. Alternate/Exception Flows

Number alternate and exception flows to indicate the step at which the variation/exception occurs. For example, a variation on step 3 is listed as 3a and a second variation as 3b, and so forth.

2a – Wireless is not able to connect to configuration computer
3a – Wireless continue to search for connection with configuration computer for a defined number of seconds or minutes.
3b – wireless turns off
4b – mode is set to standby

6. Post Conditions

- The system has received new configuration data
- The system is in standby-mode or deploy-mode, either defined by configuration settings or by mode-button.
- exception: Standby mode, in the case where it cannot connect, previous configuration settings remain

Case: Mission Deployment

1. Brief Description

Starts when: Deploy-mode is selected and entered.

Ends when: Mission is finished or aborted, and new mode is pickup-mode or error-mode

2. Actors

- User
- MCU/system

3. Pre-Conditions

- The system is configured and ready for mission
- BV is ready to be deployed in water

4. Basic Flow

19. Mode button is activated to select deploy-mode, or configuration settings selected deploy-mode
20. Status LED indicate deploy-mode
21. the mission algorithm is executed, and the various data acquisition drivers/functions are executed.
22. Data is stored in memory
23. When mission is completed data acquisition halt, and the vehicle is floated to the surface
24. The mode is set to pickUp-mode
25. status LED indicate PickUp-mode

5. Alternate/Exception Flows

Number alternate and exception flows to indicate the step at which the variation/exception occurs. For example, a variation on step 3 is listed as 3a and a second variation as 3b, and so forth.

2a – error is detected, exception is thrown, and system abort the mission
3a – Error data (cause of error, state, etc) is stored in memory
4a – The vehicle is floated to the surface
5a - The mode is set to error-mode
6a – status LED indicate error-mode

2b – Battery power level is below a certain threshold
3b – data is stored in memory
4b – Mission is aborted and vehicle floates to surface
5b – The mode is set to low-power-PickUp-mode where only the indication light is on

6. Post Conditions

Successful mission:

- Data is stored in memory
- The vehicle is floated to surface
- Indication light is on
- mode is set to pickup-mode

Exception:

- Error-data (cause of error, state, etc) is stored in storage unit
- vehicle is floated to surface
- indication light is on
- Error-mode in case of error, low-power-PickUp-mode in case of low battery

Case: Pick-Up

1. Brief Description

Starts when: PickUp-mode is selected (by software or by activating the mode-button).

Ends when: mode-button is activated and new mode is selected, or new mode is selected by software.

2. Actors

- User
- MCU/System

3. Pre-Conditions

- The mission is finished or aborted
- vehicle is floated to surface, or is out of the water
- System is powered on

4. Basic Flow

26. PickUp-mode is selected (by activated mode-button or by software)
27. Status LED indicate PickUp-mode is illuminated so that the vehicle can be identified in the water
28. The wireless is powered on and is broadcasting for User computer to connect to it
29. The user computer connects to the vehicle system
30. The user requests a transfer of the acquired data
31. Data is transferred from BV to the user computer
32. When data transaction is finished the user can select new mode from software, or by activating the mode-button.
33. Status LED indicate received status and acknowledge message is sent back to user computer
34. Status LED indicate new mode

5. Alternate/Exception Flows

Number alternate and exception flows to indicate the step at which the variation/exception occurs. For example, a variation on step 3 is listed as 3a and a second variation as 3b, and so forth.

1a – Error-mode is selected caused by an error
2a – Status LED is blinking, and is illuminated appropriately to be identified in the water
3a – The vehicle is picked up and the error is investigated by wireless communication if possible, or by serial communication.
4a – the system is reset

1b – Low-power-PickUp-mode is selected caused by battery power below a certain threshold
2b – The system is set to low-power mode, and status LED is illuminated appropriately in order to be identified in the water before the battery is empty
3b – The vehicle is picked up and battery is charged/changed in order to transfer information (or the information is transferred serially)

6. Post Conditions

- Data is transferred
- Wireless is powered off
- new mode is selected by software or mode-button

Case: Power-Off

1. Brief Description

Starts when: Mode-button is activated for a defined number of seconds (5 seconds?)

Ends when: System is powered off

2. Actors

- User
- MCU/System

3. Pre-Conditions

- System is powered on
- Data transfer is finished

4. Basic Flow

35. Mode button is activated for 5 seconds
36. System is powered off

5. Alternate/Exception Flows

Number alternate and exception flows to indicate the step at which the variation/exception occurs. For example, a variation on step 3 is listed as 3a and a second variation as 3b, and so forth.

1a – System does not respond the activated mode-button

2a – system remain in current mode until battery is empty

6. Post Conditions

- System is powered off

Appendix B

System Specification

This appendix include the system specifications, derived from the use cases in Appendix A, and form the design criteria for the embedded sensor- and control system.

BUOYANCY VEHICLE SYSTEM SPECIFICATION

Use Case – Power-On:

- The vehicle will be powered up by activating a magnetic button
- The vehicle will have indication LEDs to localize the vehicle, and to indicate the system status and mode.
- When the vehicle is initialized will default mode be standby/Idle

Use Case – Configuration:

- In configuration mode there will be established radio communication between the vehicle and a configuration computer.
- The configuration computer will receive the existing configuration settings, and be able to edit and send new configuration settings to the vehicle.
- Configuration computer interface should be able to set change the vehicle mode to standby/Idle and deploy.
- Status/indication LEDs should indicate the following: System mode, broadcast/Radio link established.

Configurable variables:

- Mission schedule 1: Depth[m], time [seconds and/or minutes]
  Mission schedule 2: Depth[m], time [seconds and/or minutes]
  Mission schedule 3: Depth[m], time [seconds and/or minutes]
  Mission schedule 4: Depth[m], time [seconds and/or minutes]
- PID: kp, ki, kd

Use Case – Mission Deployment:

- The vehicle should carry out an autonomous mission configured to dive to a certain depth, for a certain length of time.
- During mission the following data should be measured and stored: Temperature, pressure, battery voltage, movement (acceleromenter), and tilt (gyroscope). The data should be stored in flash-memory or on SD-card
- The system should be fault tolerant to a certain degree. The system should be able to detect error. In the case where software is unable to correct the error the mission is aborted, the vehicle float to the surface, and enter an exception/error mode indicated by the status LED.
- In the case where the system detect low battery voltage the vehicle should float to the surface and enter an alternative pick-up mode called low-power-pickup-mode, where only the LED is on to localize the vehicle and indicate the alternative mode.

Use Case – Pick-Up:

- The vehicle should be localized by an LED when on the surface (status LED or a dedicated localizer LED).
- Radio link should be established in order to transfer logged data without the need to pick the vehicle up from the water.
- Both (magnetic) mode button and configuration interface should be able to change system mode to: Standby/Idle, Configuration, and deploy.

Use Case – Power-Off:

- The system is powered off by entering standby/Idle mode, where a timer power the system down after an inactive period.

Exceptions/error handling:

- A status LED on the outside of the vehicle should by color code (RGB LED) or blink sequence indicate system status.
- In case of critical failure the vehicle should float up to the surface, if a localizer LED is present this should be illuminated.

- Battery voltage measured under a critical value the vehicle should float to the surface and enter low-power-pickup-mode.

Summary:

- The vehicle should have a magnetic activated mode-button that powers the system on, changes mode, and power the system off. The system is powered off by entering standby/Idle mode, where a timer power the system down after an inactive period.
- A magnetic activated mode-button should be implemented by reed-relay or a Hall effect sensor.
- The system need modules that satisfy the following requirements: Antenna, sensors, and indication LED visible on the outside of the vehicle.
- The vehicle should have several operation modes: Standby/Idle (power save), Config (configuration mode), Deploy (Mission deployment), Pick-Up (Pick up from the water and/or transfer data, re-configure and re-deploy), error mode, and low-power-pickup-mode.
- A simple configuration interface run on a configuration computer should be designed to communicate with the vehicle.
- The vehicle should send acknowledge message back to the user device when configuration is received.
- The battery voltage should be measured and logged.
- End limit switches should be installed in order to avoid damages to piston and linear actuator in case motor fail to stop in time.
- For future expansion there need to be an available serial port connection in case external sensors are added.
- As the vehicle is deployed in a marine environment with limited battery life the system should be designed for low power.
- The vehicle is designed for vertical movement between 0 meter and 50 meters below the surface.

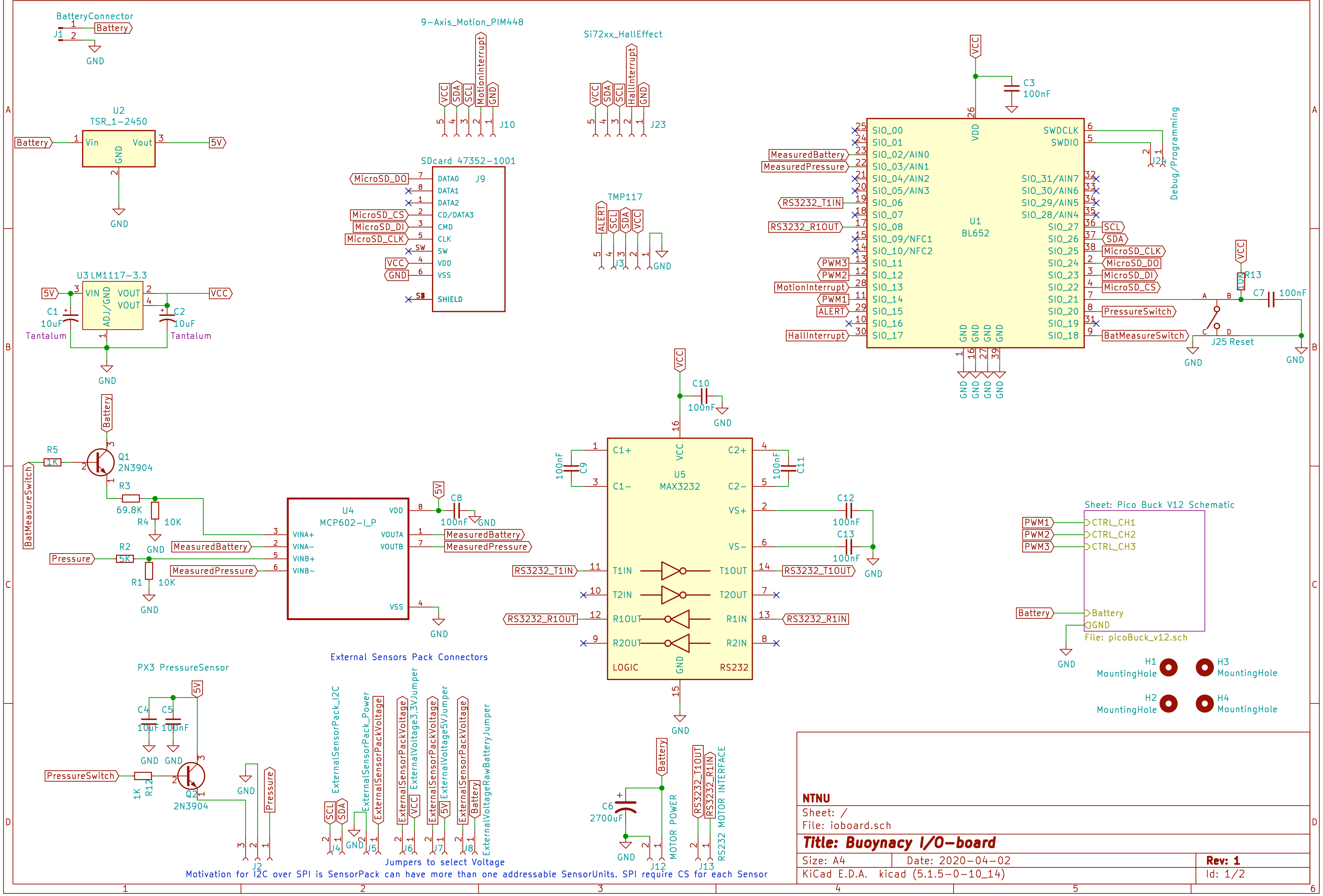
Test:

- A functional test on land to make sure the system behave in line with specifications.
- The vehicle should be tested in a water tank with no current flow or in the ocean with minimum current flow to test functionality and tune the PID controller.
- The vehicle should be tested in a large-scale sea cage to measure water current flow.

Appendix C

Schematic Revision 1

This appendix include the schematic for PCB revision 1. First page is the main page with most of the modules and components. Second page is the schematic of the PicoBuck v12 power LED driver module.



NTNU

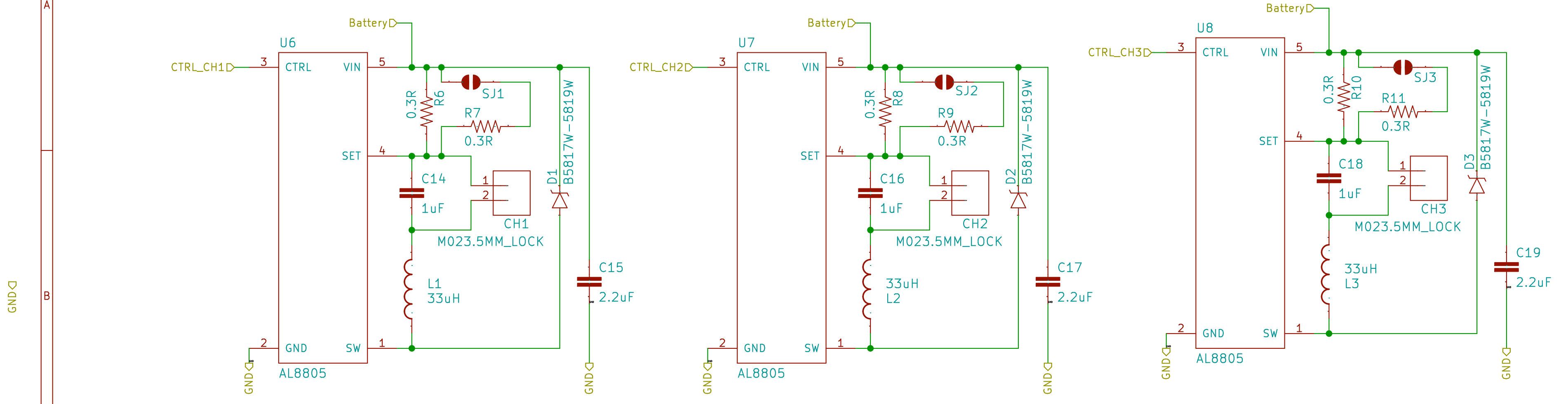
Sheet: /

File: ioboard.sch

Title: Buoynacy I/O-board

Size: A4 Date: 2020-04-02 **Rev: 1**

KiCad E.D.A. kicad (5.1.5-0-10_14) Id: 1/2



Current Sense Resistors R1, R2, R3
  - 350mA => 0.3 Ohms

Use formula: I_led = V_thd / R_set
   V_thd = 0.1

Ethan Zonca/Mike Hord   11

Sheet: /Pico Buck V12 Schematic/
File: picoBuck_v12.sch

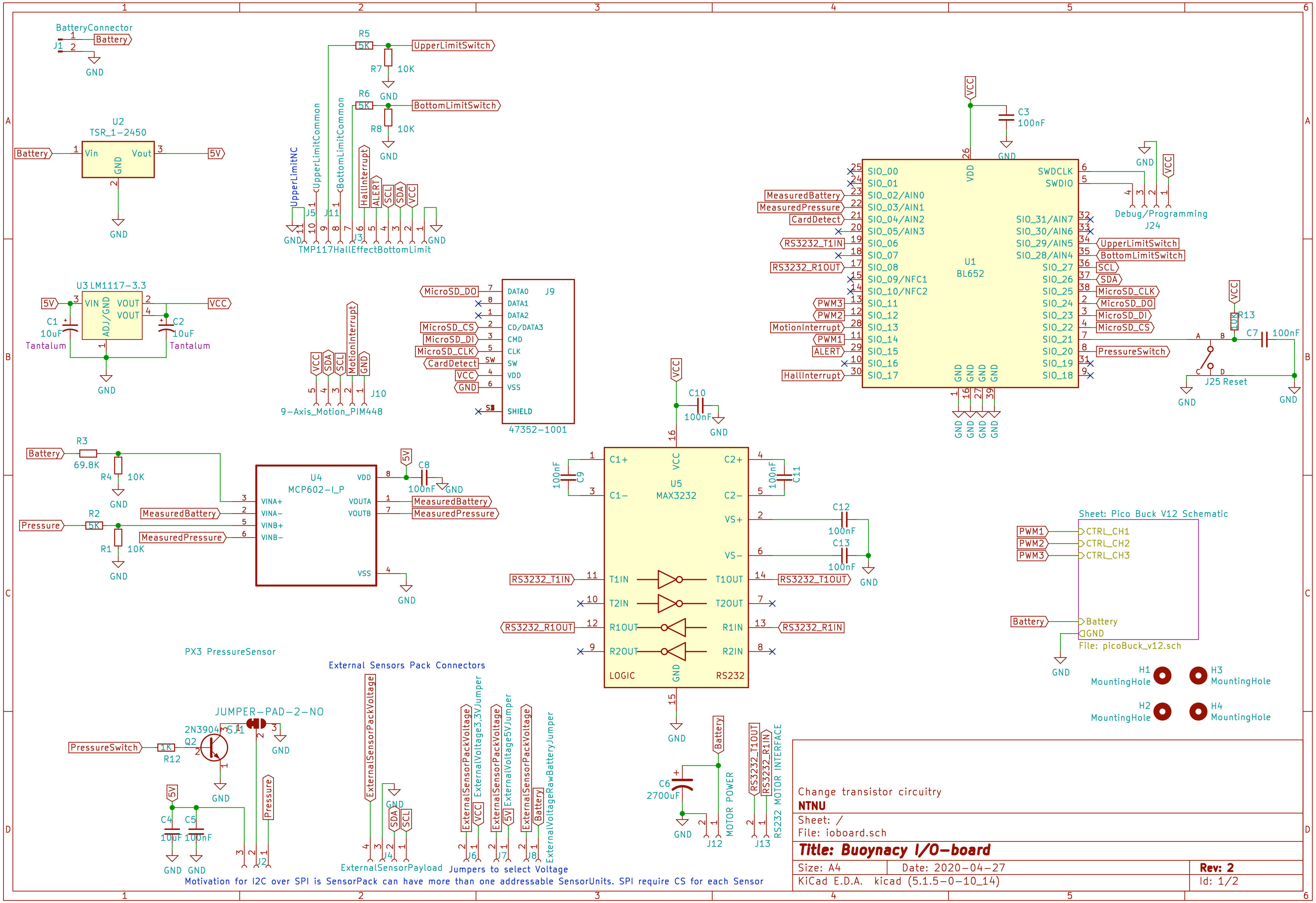
Title:

Size: User	Date:	**Rev:**
KiCad E.D.A. kicad (5.1.5-0-10_14)		Id: 2/2

Appendix D

Schematic Revision 2

This appendix include the schematic for PCB revision 2. First page is the main page with most of the modules and components. Second page is the schematic of the PicoBuck v12 power LED driver module.



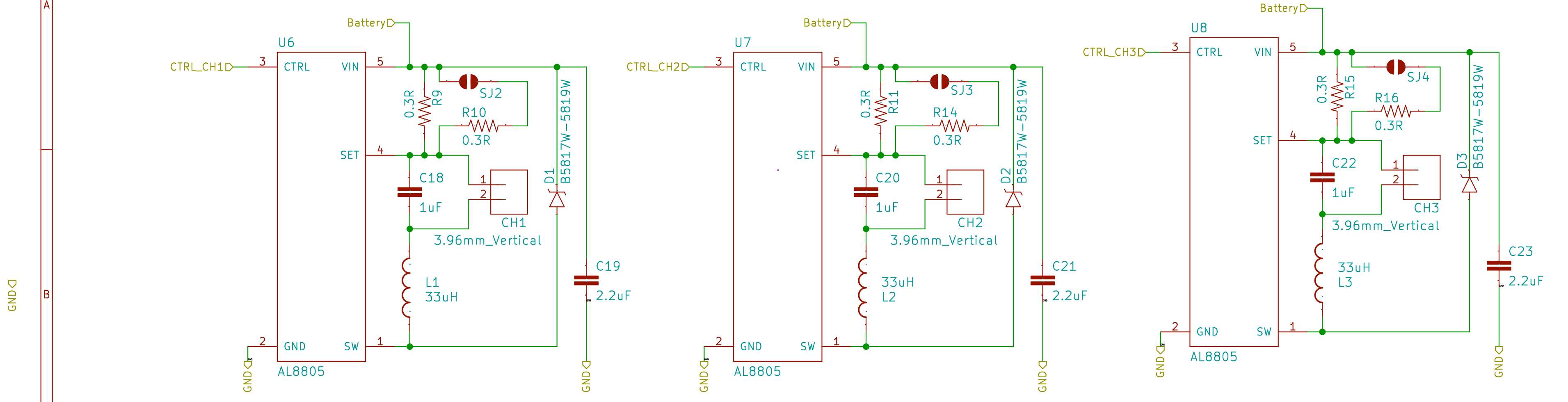
Change transistor circuitry

NTNU

Sheet: /
File: ioboard.sch

Title: Buoynacy I/O-board

Size: A4 Date: 2020-04-27 **Rev: 2**
KiCad E.D.A. kicad (5.1.5-0-10_14) Id: 1/2



Current Sense Resistors R1, R2, R3
- 350mA => 0.3 Ohms

Use formula: I_led = V_thd / R_set
V_thd = 0.1

Ethan Zonca/Mike Hord 11

Change transistor circuitry

NTNU

Sheet: /Pico Buck V12 Schematic/

File: picoBuck_v12.sch

Title: Buoynacy I/O-board

Size: User	Date: 2020-04-27	**Rev: 2**
KiCad E.D.A. kicad (5.1.5-0-10_14)		Id: 2/2

Appendix E

Maximal Vehicle Density Range

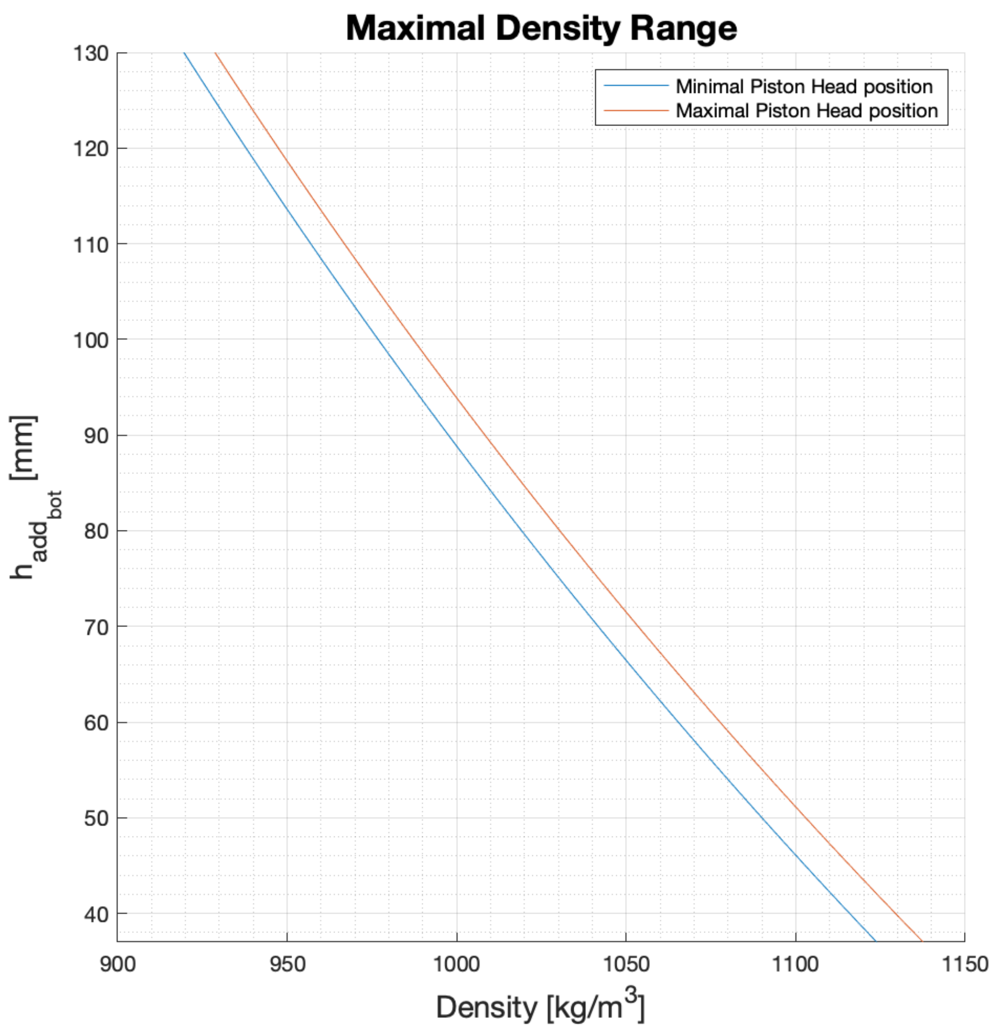


Figure E.1: Vehicle density range from minimal $h_{add_{bot}}$: 37 to maximal: 130. Density range for each Outer Lid positions are shown as Minimal Piston Head position and Maximal Piston Head position.

Appendix F

Test Plan: Functional Test

This appendix include the functional test plan. The finished system is tested on land to verify that all the core functionalities are working before the vehicle is tested in water.

Buoyancy Vehicle Test Plan

- *Test Action*
    - *Expected Behavior*

- Start program by "Reset":
    - Program start, Initialized without error, Enter IDLE state, LED dim white.
- IDLE state: To sleep state, 2 minute timer:
    - After 2 minutes in IDLE state system enter sleep state and "system off" mode.
- Wake up and reset on Hall-Effect interrupt:
    - MCU enter "system on" mode, resets, initializes, and enter IDLE state.
- IDLE state: To CONFIGURE state on Hall-Effect interrupt:
    - Transition to configure state, LED change to blue, and BLE connect.
- Use "BLE application" or "nRF UART v2.0" to navigate and set a few configuration options:
    - Configuration options are changed.
- BLE: From "Main Menu" go to IDLE state:
    - BLE disconnects and LED change to dim white signaling IDLE state.
- Use Hall-Effect to enter CONFIGURE state:
    - CONFIGURE state entered.
- BLE: navigate menu and verify configuration options hold values entered above:
    - Configuration options hold the new values from above.

Data acquisition: Upon entering MISSION state a new file, named [integer].txt - one integer greater than the previous – is created. Values of measured sensor and mission related data are written to this file during mission. One file each set of missions. Each individual mission stored in the same file, to track movement.

- BLE: go to MISSION state:

  - BLE disconnect, LED change to green, 1.txt is created, and motor start moving towards mission 1 setpoint depth.
- Use multimeter and power supply to emulate 0,5 V – 4,5 V pressure sensor voltage:
  - PID regulate motor and piston to hold vehicle in place on equivalent mission 1 setpoint depth.
- Mission finished: Transition to PICKUP state:
  - LED yellow, BLE advertising and connecting, vehicle float to surface.
- BLE: PICKUP state: reconfigure and go to mission:
  - New values are stored, and new mission begins. 2.txt is created.
- BLE: Back at PICKUP state: Go to CONFIGURE state:
  - BLE remain connected, CONFIGURE state entered.
- CONFIGURE state: Use Hall-Effect interrupt to enter MISSION state:
  - New mission begins, 3.txt is created.
- BLE: PICKUP state: Go to IDLE state.
  - BLE disconnect, LED dim white.
- IDLE state: Use Hall-Effect to enter CONFIGURE state:
  - CONFIGURE state entered, BLE connected.
- BLE: CONFIGURE state: Go to CONFIGURE state:
  - Verify nothing happens.
- Run new mission:
  - New mission started, 4.txt created.
- PICKUP state: Use Hall-Effect Interrupt to go to IDLE state:
  - BLE disconnect, LED dim brightness.
- IDLE state: Wait two minutes for sleep state:
  - Remain in IDLE state for 2 minutes, enter sleep state, LED shuts off, and enters “system-off” mode.
- Use Hall-Effect Interrupt to awaken system and start new mission. Turn down voltage to just below 12,8 V:

    - New mission is started, 5.txt is created. When System voltage is decreased to below 12.8 V mission aborts and enters LOWPOWER state. LED is dim red, and vehicle float to surface.
- Switch off power, remove SC card from vehicle, and inspect on personal computer that five mission files are created. Named 1, 2, 3, 4, and 5 respectively. Each file contains written sensor and mission related data:
    - SD card holds 5 .txt files named 1, 2, 3, 4, and 5.