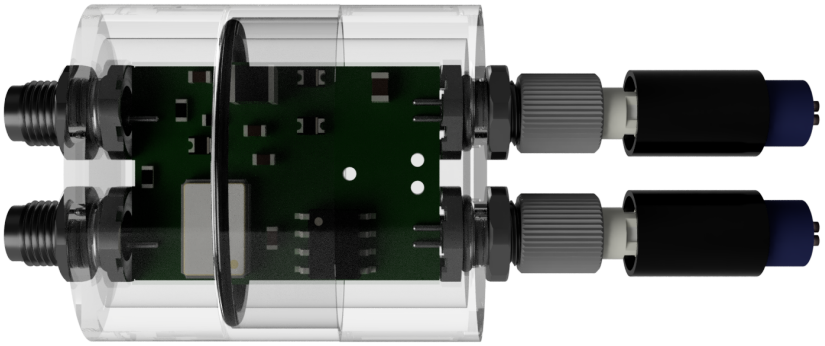
University of Southern Denmark



Semester Project PRO3

3rd Semester Robotics - 2018

Group 01

Mobile Robot Systems
Sensor Network for SDU-Vikings

Christian Eberhardt
chebe17@student.sdu.dk

Jakob Grøftehauge
jakra17@student.sdu.dk

Lars Pedersen
larpe17@student.sdu.dk

Mads Kuhlmann-Jørgensen
makuh17@student.sdu.dk

Peter Christiansen
pechr17@student.sdu.dk

Supervisor: Karsten Holm Andersen

Handover Date: 17-12-2018

Abstract

The purpose of this project is to make a sensor network for an electric race car. The sensor network will facilitate transportation of sensor data from sensors all over the car to a main controller. This requires knowledge of the disciplines: Digital signal processing, data communication and electrical design. Nodes which interface sensors are to be connected to the main controller via a single bus cable. The communication on the bus will follow the protocol of CAN 2.0A ensuring reliable communication in an environment characterised by noise induced by fast switching currents. At the core of each node is an ATmega32M1 micro controller. Nodes are designed to sample, filter and transmit data. Each node will support up to two sensors with analog outputs. A first order anti-aliasing filter is implemented on the node to allow for signal processing using a digital low pass IIR Chebyshev type II filter implemented in the micro controller. This is chosen to filter out electric noise from the environment. The anti-aliasing filter is insufficient and better solution is presented, but not implemented. As a feature the integrator is presented with the option of adjusting the cutoff frequency of the digital filter by adjusting the frequency of which samples from each of the sensors are taken. Functionality is provided that maps voltages to sensor values making it easier to interface to non-linear outputs of sensors. An extra protocol, defining a range of message types, is designed as an overlay to the protocol specified in CAN 2.0A. Each node is configured at the initial start-up of the entire network after which nodes will start to transmit data. Should a node somehow lose connection and reconnect to the network or in the event that a new node is added after start-up, individual nodes are configured to request for configuration parameters from the main controller. The network is design with security in mind ensuring that events like a sensor shorting to the supply voltage or ground, or a sensor halting transmission, are quickly flagged for the main controller.

Contents

1 Introduction

In the report some words will occur in *italic*. These words will be explained in the word list, section 15. The word will only appear in italic the first time it is presented. Cited material such as datasheets, test journals or literature will appear in square parentheses as shown here, [17]. Test journals, datasheets, source code and other material related to the project can be found in the digital appendix. A detailed description of the contents of the digital appendix is to be found in section 14.

2 Problem Description

The project is developed in collaboration with the SDU-Vikings racing team. Every year the team builds an electric racing car with the goal of competing in Formula Student competitions. In the car a suite of sensor inputs are used for operating the car as well as for post analysis of a driving session. For the sensor inputs to be reliable the sensors must be sampled properly taking into account the noisy environment of an electric racing car. The sampled data must also be transmitted to the main controller in the car for processing and storing. This could be done by connecting each sensor directly to the controller with individual wires. This would require a lot of wires and for the controller to have a physical input for each sensor. This adds extra weight and cost, as well as making implementation of new sensors troublesome. Instead, it would be beneficial to establish a network, in which all the peripheral sensors are connected to one shared bus, that allows data to be transmitted to the main controller. Figure 1 illustrates the sensors connected by a network.

The objective of this project is therefore to develop a network solution that will allow for data to be reliably sampled from the sensors and transmitted to the main controller. The solution has to accommodate different sensors with different needs regarding transmission frequency, sampling frequency, etc. and take into account the noisy environment of the electric racing car.

It is the goal of this project to offer SDU-Vikings racing team a solution, which can be implemented seamlessly in cohesion with the main controller on the car, that is currently under development in SDU-Vikings.

3 Analysis

Concurrent with the beginning of this project, the SDU-Vikings have started developing the tenths iteration of their racing car, the Viking X. Mechanically, the car is comparable to large go-karts, although it is equipped with suspension. The Viking X is 2 m long, 1.2 m wide and weighs around 200 kg. The car has a top speed of around 115 km/h, although such speeds are normally not reached during a race. The Viking X will have four wheel drive, each wheel driven by a separate motor peaking at 30 kW. However the Formula Student rule set [18] dictates a maximum power

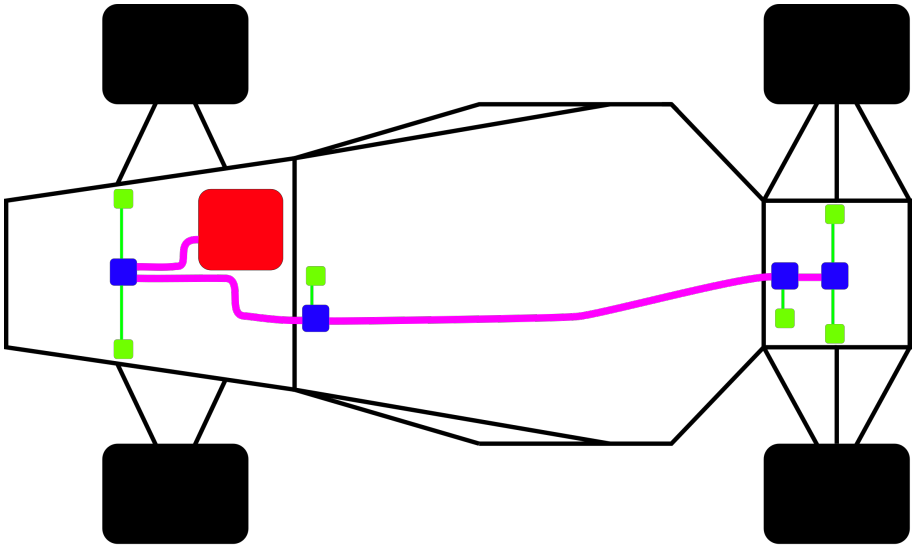


Figure 1: An illustration of the sensor network in a racing car. Sensors (green) are connected to sensor nodes (blue). The network bus cable (pink) connects all the nodes to the main controller in the car (red).

consumption of 80 kW, meaning that all four motors will not be operating at full capacity at once. This still indicates that there will be large currents running through some wires in the car. This will induce a magnetic field that can add noise to data transferring signals. In the worst case, it could lead to corrupted or lost data, meaning that measures should be taken to ensure resistance to electrical noise. This could effect the data bus as well as sensor outputs, meaning that some kind of filtering and signal processing will be desirable.

In order to excel at Formula Student competitions, emphasis is put on designing the different components in a way that minimises size, and especially weight. This should also apply to the design of the communications network. Formula student competitions take place outside on a racing track where the car may be exposed to water and dirt. Protective measures should be taken to account for this.

3.1 Signals to be Addressed

To understand the magnitude of this project, it is imperative to know, what kind of information the sensor network is required to handle. Different types of data expected to be handled by the system are listed below.

- Steering angle
  - In order to visualise the rotation of the steering wheel, and thereby the rotation of the wheels, the angle of rotation is monitored.
- Brake pressure

  - In order to analyse how the brakes perform, the pressure in the braking system is measured. The brake pressure rises when the brake pedal is pressed.

- Torque pedal position
  - The position of the torque pedal, commonly referred to as the accelerator pedal, controls the power flow to the motors, and thereby the speed of the vehicle.

- Suspension travel
  - The car is equipped with suspension. In order to analyse how the suspension system acts, especially when going through corners, the travel of the suspension pistons is monitored.

- Temperature
  - To monitor the general performance of the car, the temperature is measured at different points in the car.

In order to properly handle the data which is sampled from sensors, it is beneficial to analyse some key characteristics of the signals. Table 1 shows these characteristics. The range means the expected minimum and maximum values the signal will reach, and levels indicates how many discrete steps are needed to represent the signal with the desired resolution. The highest frequency component is to give an indication of the relevant dynamic frequency range of the signal. Transmission rate is an estimation of the rate at which the data will be needed on the main controller.

Signal	**Range**	**Levels**	**Highest Frequency Component**	**Transmission Rate**
Steering Angle	-110 - $110°$	880	7 Hz	20 Hz
Brake Pressure	0 - 150 bar	1500	< 10 Hz	20 Hz
Torque Pedal	0 - 100 %	1000	< 10 Hz	100 Hz
Suspension	-40 - 40 mm	800	< 10 Hz	20 Hz
Temperature	0 - 90 °C	900	< 1 Hz	1 Hz

Table 1: The different signal types that have to be addressed by the system.

The steering angle is the simplest signal to analyse, because a set of data from a test run of a formula student car has been provided, and can be seen is outlined in the digital appendix section 14. Although the course of the track and conditions under which the data was collected is unknown, it does give a good indication of what characteristics to expect from the signal. Unfortunately no data has been available for the other signals, but the steering angle data will be used as a reference for the brake pressure and torque pedal, since both of these are also manipulated by human interaction.

The steering angle data provided by SDU-Vikings is shown in figure 2. From this data it can be extracted that the maximum angle reached is $107°$ and the minimum is $-85°$. It is expected that in general the signal range will be symmetric around 0, thus -110 to $110°$ is chosen. The steering angle data is not used for controlling the car, but merely for visualisation purposes. Therefore it

has been established that one quarter of a degree of precision is sufficient, resulting in 880 steps within the range.

To analyse the frequency components of the signal, a single sided amplitude spectrum is generated using the fast Fourier transform. The amplitude spectrum is shown on figure 3, which indicate that no signal components of significant magnitude appears above 7 Hz. The transmission rate is chosen in a way that yields at least ten samples for the rise time of an edge. The shortest rise times for the signal are between 0.5 s and 1 s, resulting in a transmission rate of 20 Hz. The same rate is chosen for the brake pressure, as similar characteristics are assumed to be expected from another signal related to human interaction. The same argumentation is used for the highest frequency component of both brake pressure and torque pedal, however to be safe they have been estimated to be less than 10 Hz. The range of the brake pressure has been defined by SDU-Vikings, and if expecting a precision of 0.1 bar, 1500 levels are required.

The signal from the torque pedal is used to control the car, which leads it to be a system critical signal described in section 3.2. Because of this the transmission rate is set to 100 Hz. The position is expressed as a percentage. It has been estimated that 0.1 % is a high enough resolution to represent the travel of the pedal.

The suspension of the car allows for the chassis to move independent of the wheels. For instance when going through corners. Each of the suspension pistons can travel ±40 mm from their neutral position. It is estimated that a resolution of 0.1 mm is sufficient, resulting in 800 levels. Since it is far beyond the scope of this project to conduct calculations on the mechanical properties of the car, the frequency components and transmission rate are based solely on information received from, the SDU-Vikings mechanical team [35].

The range of the temperature sensor is based on the temperatures expected in the hottest areas of the car. Rule EV 4.5.3 [18], states that all *high voltage* system wiring and parts are to be able to withhold 85 °C. A maximum threshold of 90 °C has been selected to allow some margin of error. The car will not operate in sub-zero degrees and therefore the range of 0 - 90 °C has been chosen. Based on the limited resolution of common temperature sensors, it has been estimated that 0.1 °C is suitable for this application.

The highest frequency component and transmission rate are based on two cases where temperature is measured: On the discharge resistor and in the cooling system. The discharge resistor is used when the car is powered of to discharge the large capacitors in the inverters by heat dissipation. The cooling system is water based and used for cooling the motors and inverters of the car. Water has a high heat capacity resulting in a slow change of temperature. The discharge resistor has a relatively large mass and is designed specifically to transfer heat to its surroundings. Therefore the change in temperature for this piece is also expected to be very slow. Based on these assumptions, the highest frequency component of the temperature signal is expected to be less than 1 Hz, and transmission rate can be as low as 1 Hz.

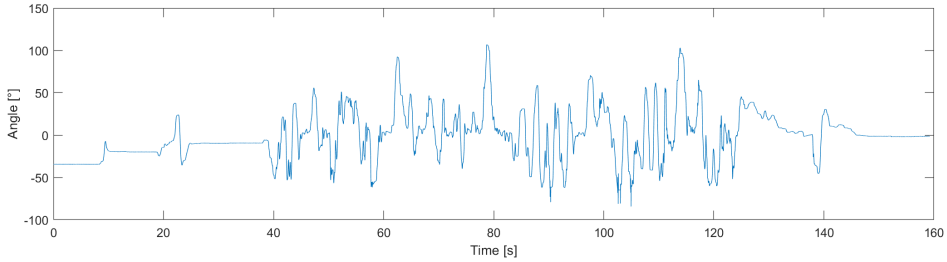


Figure 2: Plot of the Steering Angle data.

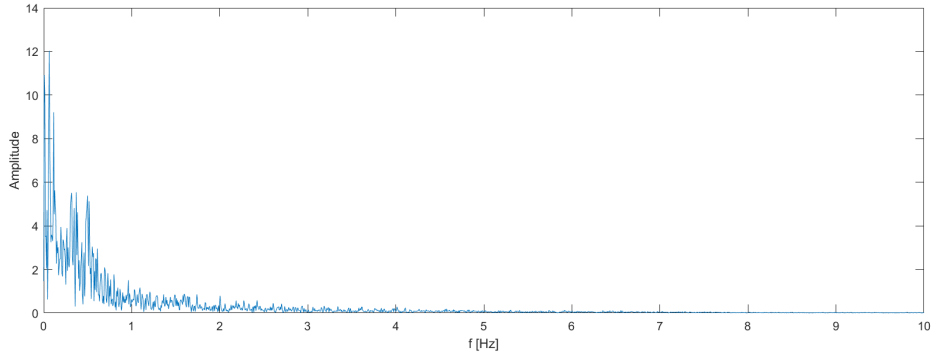


Figure 3: Single sided amplitude spectrum for the steering angle data plotted from 0 to 10 Hz.

3.2 System Critical Signals

Some signals are critical to the operation of the vehicle, as they feed directly to the main controller of the car and influences the logic used to control the car during races. These signals are called system critical signals, abbreviated *SCS*, and are signals that influence the *shutdown circuit*. SCS are described in rule T11.9 in the Formula Student rule set [18], which dictates that if an insoluble failure is detected on an SCS, the car is to enter safe state. As mentioned, for the scope of this particular project, one sensor that caries an SCS is of interest; the torque pedal position sensor. More SCS sensors may be needed in the future.

Since these signals are critical to the operation of the car, if a sensor encounters an error, or if transmission of such a signal suddenly halts, it must be detected and acted upon in due time. The Formula Student rule set does not specify such a time frame. To establish a suitable time, the inverters, which are very critical as they supply power to the motors, have been used as a guideline. Looking at the datasheet for the inverters used in the Viking X [13], it has been found that they have a failure mechanism which responds if no update from the main controller has been received for 50 ms.

3.3 Possible Network Solutions

When transmitting data serially on a network there are lot of things to take into consideration such as how to define the physical layer, do error checking, handle addressing, decide priorities etc.

Start Flag	Address	Function	Data	Error Check	End Flag
28 bits	8 bits	8 bits	n · 8 bits	16 bits	28 bits

Table 2: An illustration of a Modbus frame using the RTU transmission mode.

SOF	ID	Control	DLC	Data	CRC	ACK	EOF
1 bit	11 bits	3 bits	4 bits	0-64 bits	16 bits	2 bits	7 bits

Table 3: An illustration of a standard CAN-frame. SOF means start of frame, DCL means data length code, EOF means end of frame and ACK means acknowledge.

Although it is possible to design the network from scratch, there also exist a plethora of predefined network and protocol standards to choose from. The ones examined for this project is listed below.

- I^2C
- SPI
- RS-232
- RS-485
- Ethernet
- CAN
- Modbus

As described in section 3 the physical environment where the network operates is characterised by a large amount of electrical noise. Therefore it is important that the chosen solution is robust and resilient to electrical noise. I^2C, *SPI* and RS-232 all utilise single ended signals to transmit data [6] [10] [9]. This means that one wire carries the signal as a voltage and another wire acts as a reference, typically ground. This method is quite vulnerable to electrical noise and therefore these solutions are not suitable for the project and will not be discussed further.

RS-485 is a definition of a physical layer where data can be transmitted via differential signals. Which means that information is carried via two wires with complementary signals mirrored around a midpoint voltage [5]. This makes the solution much better suited for a noisy environment. However, since RS-485 only defines the physical layer, it is necessary to combine it with a communications protocol that defines framing, addressing, error control etc. Modbus constitutes such a protocol. Modbus is widely used within the automation industry and defines a master/slave system where one master node sends requests to the slave nodes that will reply to the request. Data is transmitted in variable sized packets that contain six different sections as illustrated in table 2. Start and end flags define the beginning and end of the frame. An address field that contains the address of the requested node, and a function field that defines which operation the node should perform, e.g. return contents of a register. A data section holds information about the operation, e.g. which registers to read, and in the reply telegram it holds the data to be returned. Lastly, the protocol also brings means of error control by adding a 16-bit *CRC* [7].

CAN offers a lot of the same utility that the Modbus using RS-485 does. It utilises differential signals and has a definition for framing, see table 3, as well as a range of error checks. The exact workings of these utilities differ from that of Modbus, but the most significant difference is that CAN defines a master/master system. This means that all nodes can transmit data at will as long as the bus is idle rather than having to wait for a request from the master. Each message on the

network is assigned a unique identifier, which is used in an arbitration phase to determine which node can communicate on the network. CAN was developed for the automotive industry and is an international standard for an in-vehicle network known as ISO11898 [22] [15] [16]. This standard defines that the differential signals shall be transmitted in cables with a characteristic impedance of 120 Ω and the bus must be terminated with 120 Ω resistors at both ends [33, p. 2,9] [3]. It is common to transmit the CAN signals via twisted wire pairs in a shielded cable in order to make it more robust towards electromagnetic noise. It is also preferred to accommodate voltage supply and ground using the same cable as the CAN signals.

Ethernet is a protocol which is mostly used for transmitting large data packages. The standard version supports a payload between 46 and 1500 bytes per transmission and is capable of transmitting data at high speeds, some versions upwards of 10 Gbit/s. Ethernet is intended for large networks consisting of numerous nodes. It is designed with an overhead size of 18 or more bytes depending on the version. For the standard version, 6 bytes is allocated for both the source address and destination address [17]. The Ethernet protocol is deemed too complex for the relatively small sensor network, and has not been examined further.

4 Delimitation

As introduced in section 3 the project presents a wide range of challenges and possible solutions. Some solutions might be equally valid, but may offer different approaches to the project. This section will define the approach chosen. It has been decided that it will be of high priority to deliver a functional product to SDU-Vikings by the end of the project. This provides the basis for the choices presented in this section.

4.1 Network

From section 3 it appears that using CAN, Modbus or a custom designed communications protocol via RS-485, could be equally appropriate for data communication. This is because the data which has to be transmitted fits within the respective payloads, while the overhead is reasonable. It has been decided to put considerable effort into designing and adding functionality to the nodes that will interface the sensors. This rules out designing a custom communications protocol from scratch, as designing and implementing the protocol is very time consuming. CAN is also preferable to Modbus since CAN offers a more finished solution, which aligns with the main priority mentioned previously. Lastly CAN is widely used in automotive and automation industry and is therefore an interesting prospect from the view of a robotics engineer. Therefore it is chosen to use CAN 2.0A with a bit rate of 1 Mbit/s for data communication.

4.2 System Configuration

Figure 1 shows how the network is intended to be implemented in the car. Nodes distributed throughout the car will monitor sensors and transmit data to the main controller via a shared bus. The primary focus of the project is developing the nodes. Since the sensor nodes are designed for a race car, they need to be as small and lightweight as possible only containing functionality required for interfacing with sensors, processing, and transmitting sensor data. It has been chosen to build the nodes around an AVR micro controller as experience from previous projects can be utilised. Ideally several sensors can be interfaced using the same node, for simplicity this number has been limited to two. Also for simplicity it has been chosen to only support sensors with analog voltage outputs between 0 and 5 V.

To control the nodes and receive the data that they transmit, a task must run on the main controller. The main controller is not available during the project, so instead it has been chosen to reprogram one of the node boards to take on the role of the aforementioned task. This will be referred to as the hub. The functionality of the hub must later be ported to run on the main controller, but that is beyond the scope of the project. This also means that it will not be explored how data can be logged and saved in the main controller. Only the newest data from each sensor is to be kept in memory.

It has been chosen to focus as much of the functionality as possible on the node in order to minimise the computation time of the main controller. This means that all signal processing and filtering will take place before the data is transmitted to the hub. At the same time, the system will operate as a master/master system where data will be transmitted from the nodes at defined intervals.

4.3 Programming

Working with a relative small embedded system, some limitations to the processing power of the system are present. Although no previous experience with C is present in the group, it is decided to use C to write the software for this project, as C is the most widely used language for developing embedded software, due to the efficiency of the language [2]. Furthermore a library for CAN written in C code is available for AVR the micro controller. Though it was later discovered that the library was not usable for this system, and adaptions had to be made.

4.4 Data Processing

As mentioned in section 3, it is wanted to digitally filter the data sampled from the sensors. When sampling the signal from the sensors, an analog anti-aliasing filter must be used in order to avoid aliases of higher frequency components. For simplicity a first order RC filter has been decided on as an anti-aliasing filter. This has been discovered to be a remarkably poor choice, and the implications will be discussed in section 11. The requirements for the anti-aliasing filter depend on the rate at which the signal is sampled. It has been chosen that sampling will occur no faster than at 500 Hz and no slower than 100 Hz.

The purpose of the digital filter is to filter out noise on the signals from the sensors. It is expected for the noise to be fairly high frequency signals, such as voltage spikes. Since the signals from the sensors, as shown in table 1 are expected to be quite low frequency, it is assumed that only low pass filters will be of relevancy. Since the values shown in table 1 are based on a lot of estimations and therefore might not be completely indicative of the actual signals, it is desirable to allow for filters with different cutoff frequencies to be selected amongst.

It has been decided to only support sensors which output voltages between 0 and 5 V, however, the desired signal probably has a completely different range, and may not be linearly related to the voltage range. For instance, the desired output of a temperature signal, as described in section 3, is expected to be in the range of 0 and 90 °C. By providing a polynomial which describes the relation between the input voltage and the desired output signal, the node can administer the conversion before transmitting the data to the hub.

If the signal processing described in this section is to be carried out with sufficient precision, it cannot be done using only integers. In order to overcome this issue, real valued numbers can be represented using either a fixed point format or a floating point format. Out of convenience it has been chosen to use the floating point format for all real valued numbers used for the project. Since calculations using this format are fairly computationally heavy, it must be ensured that the calculation of the filter does not take up more than 50 % of the computation time.

5 Requirement Specifications

Based on section 3 and 4 a series of requirements, which are presented below, have been settled upon for the project. Unfortunately, some of the requirements will not be formally tested due to limitations in time and equipment available. Although not all requirements can be tested, they are worked towards as if they could. Requirements that will be tested are written in **bold** text.

- **The network system must be able to start up successfully within 100 ms. Successful start-up is confirmed when all entities on the network have verified their initialisation.**
- **A maximum bus load at which the operation of the system is guaranteed must be determined. At this load 1000/1000 messages must be received by the hub, and no messages may be postponed by more than what is equal to transmission of two messages.**
- **The hub must give an alert, within 50 ms, if transmission of messages related to an SCS is halted.**
- Each node must be fitted in suitable casing which protects it from water and dirt.
- Each node must be able to interface two sensors with output signals of 0 to 5 V.
- **Each node must be able to detect so called *wire-off* errors, if the input from a sensor goes below 0.5 V or above 4.5 V.**

- **Each node is to be able to request setup instructions upon connecting to the network.**
- Each node must have a function which, based on a provided polynomial, can convert sensor input signals from 0 to 5 V to the desired format.
- An anti-aliasing filter with a cutoff frequency of maximum half of the minimum sampling frequency must be fitted on the node.
- Digital low pass filters must be implemented on the node in order to filter out high frequency noise. It must be possible to choose between at least three different cutoff frequencies.
- **At a sampling frequency of 500 Hz, execution of digital filters for two sensor inputs may use no more than 50 % of the CPU time.**
- The bus cable must be shielded and house two sets of twisted pair wires to accommodate *CANH*, *CANL*, 5 V supply and ground.

6 Hardware

Among the most important decisions in the design process of the physical node is the choice of micro controller and other vital components. Solutions have to be found in order to comply with the overall requirements listed in section 5 and to design a circuit board layout, that allows for a small physical node.

6.1 Network

Figure 4 shows the network topology for a network consisting of N nodes with twisted and shielded cables. Each node is capable of connecting up to two sensors. Each node is daisy chained to the next therefore requiring each node to have two connectors for connecting to the bus. At the ends the bus will be terminated with 120 Ω resistors. The figure shows the main controller of the car positioned at one of the termination ends however the position of the main controller can be arbitrarily chosen.

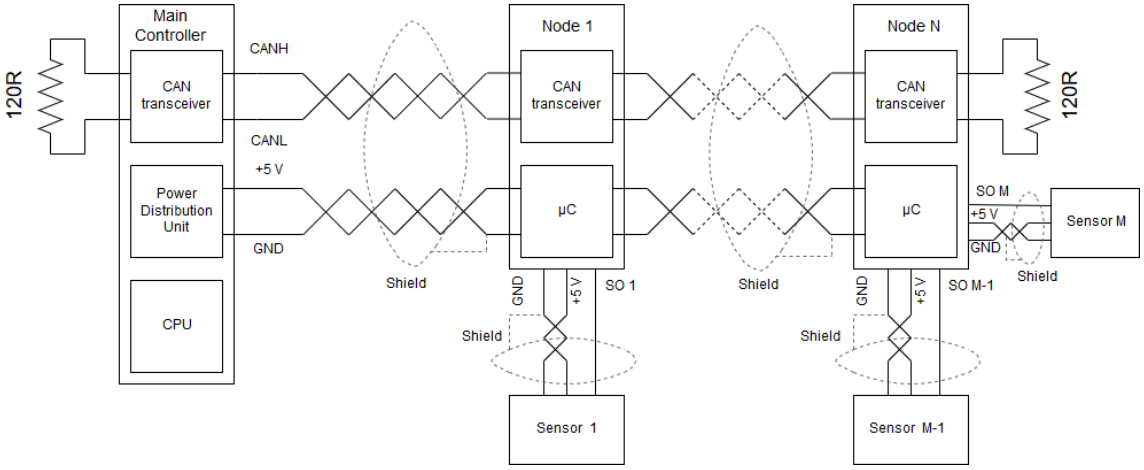


Figure 4: Concept diagram for network of nodes connected to the main controller of the car. SO is Sensor Output.

6.2 Cables and Connectors

When it comes to finding the right way to connect the nodes to the network and connect sensors to the nodes, there are a wide range of options. Each has advantages and disadvantages. It is ideal to find a solution which can be used for both purposes. Table 4 gives an overview of some solutions that have been discussed.

Ultimately, the circular connectors have been chosen as the preferred solution due to the easy and secure connections that it offers, while at the same time being weather proof and having a relatively small footprint on the *PCB* [29] [28] [27] [26]. Figure 5 shows how the PCB fits perfectly between the male pins of the connectors. This is more clearly visible in the digital appendix where a render of the PCB can be found. A drawback to this solution is that adding termination resistors is non-trivial. This issue can be solved by designing a special termination piece consisting of a 120 Ω resistor connected to the connector piece.

The cable chosen for connecting nodes is a shielded bus cable with a characteristic impedance of 120 Ω, a diameter of 5 mm. The cable contains two twisted wire pairs meeting the requirements of a twisted wire pair for CANH and CANL and a twisted wire pair for supply voltage and ground [24]. The connectors are soldered manually to the cable connectors in each end, which feature soldercups. A layer of heat shrink is applied to the solder joint to ensure that the joint is water proof as seen in figure 5.

6.3 Node Casing

As mentioned in section 3, the car will be operating outside and therefore all components have to be protected against water and dirt. Because of this, it is necessary to come up with some kind of casing that protects the circuit board. Several different solutions have been discussed, the three

Solution	Advantages	Disadvantages
Soldering the cables directly to the PCB.	• Removes the need for choosing connectors. • Makes for a very tight PCB design.	• Very inflexible - making changes to the network is very cumbersome. • Changing the sensors connected to the node is very cumbersome.
Micro USB connectors.	• Easy connection to the network as well as sensors.	• Takes up considerable space on the PCB. • Does not offer a way to secure the connector. • Difficult to make weather proof.
Terminal block plug connectors.	• Easy connection to the network as well as sensors. • Easy to implement termination resistor in the connector.	• Takes up an extreme amount of space on the PCB. • Potentially difficult to make weather proof.
M5 Circular connectors.	• Easy connection to the network as well as sensors. • Weather proof. • A secure connection is possible by screwing the connectors together.	• Weatherproofing the cable connector is difficult

Table 4: Possible solutions for connectors for the node.

Solution	Advantages	Disadvantages
Casting the node in epoxy.	• Requires no mechanical design. • Guaranteed to be waterproof. • Visible LEDs	• Disassembling the node is impossible. • Reprogramming the node is impossible.
3D-printing a case.	• Designing a case that can be disassembled allows for the node to be reprogrammed. • Can be designed in a way that makes LEDs visible.	• Requires considerable mechanical design work. • Can be hard to make completely sealed.
Wrap the entire node in heat shrink.	• Little to no mechanical design work is required.	• Reprogramming of the node requires breaking the casing. • Difficult to make completely sealed, especially if several sensors are to be supported.

Table 5: Possible solutions for the protective casing for the node.

best suited are presented in table 5. It was chosen to design and 3D-print a casing, which can be disassembled, since it allows for easy access to the node, if reprogramming is needed.

6.4 Node Electronics

For the choice of micro controller, it was preferred to use one from the AVR family, due to previous experience with these. Since CAN was chosen as the protocol for transmitting data on the network, the chosen micro controller must support CAN. The micro controller must also be equipped with an AD-converter with at least two inputs to support two sensors. A micro controller with well documented libraries for CAN functionalities is preferred. The *ATmega32M1* was chosen because it fits all of these criteria. It has an AD-converter with a resolution of 10 bits, which is sufficient to represent all but one of the signals, which are expected to be handled by the network. The resolution of the brake pressure signal will be lower than initially desired. Other important components include an oscillator, CAN-transceiver and anti-aliasing filters on the inputs of the AD-converter pins.

In order to transmit via CAN at the relatively high baud rate of 1 Mbps, it is important that the clock frequency of the micro controller is very exact and consistent. In the datasheet for the ATmega32M1 it is recommended to use an external crystal/oscillator as the internal oscillator is too unreliable [14, p. 240]. An oscillator can be constructed using a crystal and two capacitors, although, to save time and prioritise other aspects of the project, a premade oscillator of the type Aker Series 7 clock is used [12].

Although the ATmega32M1 has a built in *CAN controller*, it cannot supply the differential signals which will carry data on the network. Therefore the data must be sent via single ended signals to a CAN transceiver, in this case an IFX1050GVIO [21], which converts the single-ended signals to differential signals that comply with the CAN specification and ISO standard.

A simple RC filter only consisting of a resistor and a capacitor has been chosen and are placed on the node as an anti-aliasing filter. To achieve a cutoff frequency of 50 Hz the resistor and capacitor values shown in equation 1 are chosen.

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 31.973\,\mathrm{k\Omega} \cdot 100\,\mathrm{nF}} = 49.78\,\mathrm{Hz} \quad , \tag{1}$$

where f_c is the cutoff frequency given in Hz, and R and C is the resistor and capacitor value respectively [1].

6.5 PCB-design

To make the node as small as possible, the PCB must be designed to be as small as possible. The circular connectors dictates the lower limit of the PCB size, since there must be space for fastening the connectors. Making an incredible small board is highly inconvenient as extra pins and LEDs are essential while developing. Therefor a development board, for which the size is not essential, is made to be used while developing and testing.

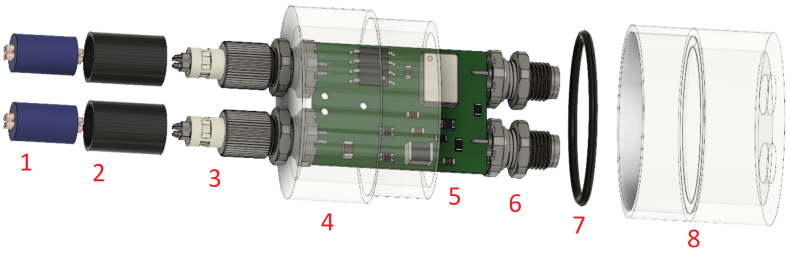
When deciding which pins on the micro controller to route, most of them do not require much thought as they are mandatory for the operation of the node. These include supply voltage, clock source, CAN signals, etc. The complete circuit schematic, which shows all routing, can be found in the digital appendix which is outlined in section 14.

Thoughts have been put into designing the routing layout in order to ensure reliable performance. The traces of CANH and CANL on the board are kept directly on top of each other for as much of the way as possible. Decoupling capacitors are added to every active component on the board to minimise noise. The trace connecting the output of the oscillator to the clock input pin at the micro controller is kept as short as possible. Decoupling capacitors are kept as close as possible to the the integrated circuits they decouple. There is a ground plane and no signal traces underneath the oscillator to ensure the oscillation with minimal deviation. In each side there are two connectors. On the CAN-connector side there are two 4-pin M5 circular connectors for connecting the node to the CAN bus. On the sensor-connector side there are two 3-pin M5 circular connectors totalling the amount of connectors on the board to four.

Programming and Debugging Interface for Node

The node is programmed trough a Serial Peripheral Interface (SPI) in the micro controller using In-System Programming (*ISP*) devices. The RESET pin in the micro controller is routed for use

when debugging via *debugWIRE*. Standard ISP-devices that have proved to work on the node are Atmel-ICE, which supports debugging through debugWIRE, and USBasp v2.0, which does not. On the development board a standard 1.27mm pitch *IDC*-connector is used. On the node intended for integration a Tag-Connect hand-held connector is used since it has a much smaller footprint than the IDC-connector and requires no bulky connector housing.



(a)

No.	Part
1	CAN bus cable
2	Heat shrink
3	M5 IP67 cable connector
4	Inner casing
5	PCB
6	M5 IP67 panel connector
7	O-ring
8	Outer casing

(b)

Figure 5: 3D-model of the node assembly (a) and annotation and explanation (b).

6.6 Conclusion

To connect nodes on the network a shielded cable containing two sets of twisted wire pairs has been chosen. This accommodates both CANH and CANL as well as supply voltage and ground. The cable has a characteristic impedance of 120 Ω, thus complying with the stated requirement. An ATmega32M1 has been chosen as the micro controller. The size of the circuit board has been kept to a minimum dictated by the connectors. M5 circular connectors are used for connection to the network bus as well as for connection between node and sensor. A protective casing has been designed and 3D-printed to protect the node from water and dirt. The final product is seen on figure 5, which summarises the different hardware choices made, to accommodate the requirements.

7 General Structure of Network Solution

As mentioned in section 4, the core functionality of the system is for the nodes to transmit data to the hub at defined intervals. Information on how to set up transmissions as well as the signal processing, for each of the attached sensor, must be supplied to the node. It has been chosen to address the sensors individually, meaning that even though two sensors can be attached to the same node, from the perspective of the network, they have no relation. The node is merely seen as an intermediate link connecting the sensors to the network. In the current and later sections, when referring to a sensor, what is meant is the instance of a sensor on the network. Since the hub

must be knowledgeable of all sensors on the network, it has been decided that the hub will also be responsible of providing information on how each shall be configured. The information related to the layout of the network must be supplied to the hub by the *integrator*. The system assumes the provided information to be correct as it has no mechanism to validate it. When the network is powered on, all sensors are configured to a default configuration where no transmission occurs. It is then the responsibility of the hub to set up each of the sensors. Once all sensors are configured, transmission will occur automatically and the responsibility of the hub is to receive the transmitted data. In section 3 it has been introduced that if data transmission halts from a sensor carrying an SCS, an alert must be triggered. To meet this requirement it has been decided to implement a so called heart beat monitor that will respond if no message has been received from a sensor within a given time period. It has been chosen to monitor both regular and system critical signals, however the time period is much longer for regular signals. The hub has the information on which sensors carry SCS and thus can distinguish between them.

The micro controller offers no options of using an operation system for easy scheduling of tasks. Therefore if switching between tasks is required, the functionality has to be implemented into the program structure itself. The developed system has the needs to schedule tasks with a specified period. As the sampling and transmission of data needs to execute with a fixed period defined by the hub at start-up. Scheduling of tasks has been implemented by having an interrupt on a timer triggering for every 0.5 ms. Every time the interrupt is triggered, the main functionality checks if any of the tasks are scheduled to run within the particular time period. If a task is scheduled to run it is executed before it is checked if other tasks needs to be scheduled. If too many tasks are scheduled to run within a time period the tasks are just postponed and fills out the CPU time of the following time periods.

7.1 Polynomial

As described in section 4 the data from the sensors must be converted into the desired format before being transmitted to the hub. To do this it has been determined to implement a function which, given a polynomial that describes the relation between the voltage input and the desired output, can carry out a conversion. The polynomial will have the form shown in equation 2, where a_1 through a_8 are the coefficients which will have to be supplied to the sensor. The program supports a polynomial of up to seventh order, however it is expected that most relations can be described using an order of three or less. The default configuration of the sensor has $a_2 = 1$ and all other coefficients set to 0.

$$f(x) = a_1 + a_2x + a_3x^2 + \ ... \ + a_8x^7 \tag{2}$$

7.2 Hub

The hub is considered the controlling unit in the network in the sense that this unit is the only one capable of setting up and shutting down sensors connected to the network. The main structure of the code consist of a setup section, a main section and an interrupt section.

The setup section is responsible of setting up all the sensors connected to the bus. Prior to start-up the hub needs to know all information regarding each sensor in order to setup the entire network. For storing this data a structure containing all information concerning one sensor has been defined. Each sensor is represented by an instance of the structure on the hub. These structures are stored on a list allowing for easy accessing of data on sensors. When the setup section is executed the program is indexing through the list of structures checking if a sensor has confirmed a successful setup. If no previous confirmation of a successful setup of the sensor has been received, the hub attempt to setup the sensor. If the hub has tried to setup a sensor a total of three times, additional attempts will not be tried.

After the initialisation of the network the main section is entered. The main section consist of a loop executing for ever 0.5 ms implemented with the principle described in section 7. The heart beat monitor mentioned in section 7 is executed inside this loop, checking SCS every 25 ms and regular signals every 125 ms. The period of 25 ms for SCS is chosen in order to have a maximum response time of 50 ms.

The interrupt section is responsible of receiving and decoding messages. Meaning code within the section is executed every time a message is send to the hub. The interrupt code acts differently depending on the received message type. For instance if a data message is received the logic is responsible of transferring the incoming data to the structure containing the information regarding the particular sensor. The other cases are illustrated in figure 6. It is acknowledged, that it is not good practice to add too much functionality within an interrupt, since it will delay the general execution and make timing unpredictable. A test, described in the journals [11, p. 8], has been conducted to estimate the magnitude of the issue. Upon receiving a setup request, the interrupt execution takes 4.224 ms, which is a very long time, otherwise the execution times seem reasonable. This event will only occur if nodes, carrying new sensors, are connected while the rest of the network is already running. Since this is a rare occurrence, the issue is deemed less critical, however it is discussed further in section section 11. Despite the possible issues, it has been chosen to do it this way as it was quick to implement. The chosen solution also eliminates the possibility of *race conditions*. The time spent in the interrupt is tested in section 10 to investigate how much CPU time is spent in the interrupt.

7.3 Node

As shown in figure 7, the node has two responsibilities, sampling data from sensors and transmitting data to the hub. Sampling data includes applying a digital filter and transmitting data includes converting into the desired format, e.g. temperature from 0 to 90 °C. Using the technique described in the beginning of this section these two functions are scheduled to run at fixed intervals defined by the configuration received at start-up. During the first 100 ms after being powered on, the node will wait for setup instructions from the hub. If no setup instructions are received, the node will send a message to request a setup. Only if setup instructions are received by the node it will begin sampling and transmitting. Otherwise it will remain in the default setup which is to never schedule sampling and transmission. Setup instructions are expected to come in at start-up, but can be

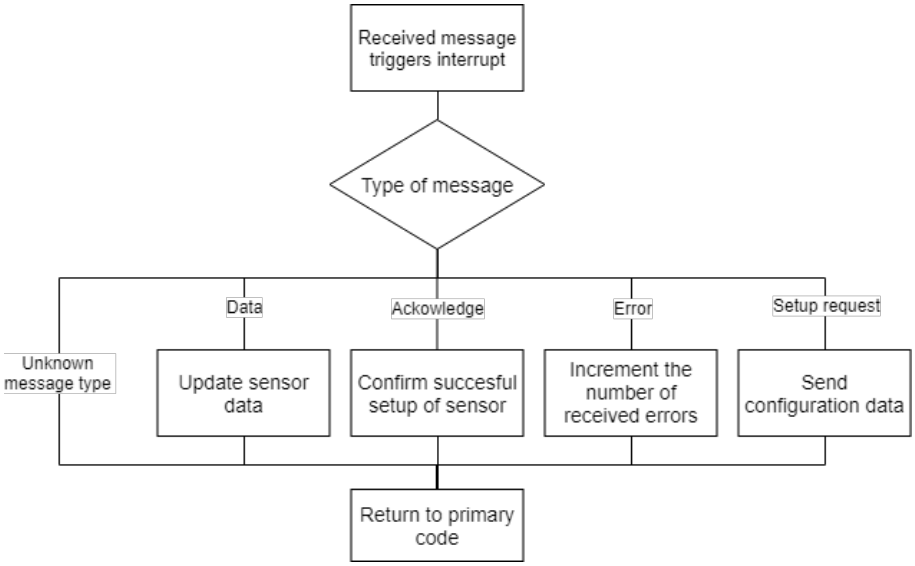


Figure 6: Flowchart of the interrupt section on the hub.

handled at any time allowing to change the setup of a node while the system is running.

Reception on the node is implemented such that upon receiving a message, an interrupt is triggered. During this interrupt the received data is copied into memory and a flag is set indicating that new data is available. The data is then decoded and dealt with during regular execution, which introduces a risk of a race condition discussed in section 11. The flag indicating new data is continuously checked during execution.

7.4 Libraries and Drivers

In order to share functionalities between different devices, and to make porting to a different system easier, it is regarded as good practice to keep different kinds of functionalities separated into different program files, referred to as libraries. Furthermore it is beneficial to separate code, which is specific to the particular hardware used, into driver files. As an example, programs for both the hub and the node utilise a timed interrupt running at 2000 Hz. Therefore, the commands for setting up the timer and activating the timed interrupt are saved to a separate library file rather than writing it directly in the main code. This way the same setup can easily be included for both the hub and the node. Then, if a change has to be made, e.g. changing the interrupt frequency to 4000 Hz, only the library file has to be edited. Which registers are to be configured, and how, in order to setup and enable a timed interrupt can vary a lot depending on the hardware used. Therefore, the specific setup used for the ATmega32M1, has been separated into a driver file.

7.5 Conclusion

The system has been structured in a way where each sensor connected to nodes on the network are treated as individual network entities. It is the responsibility of the hub to supply configuration

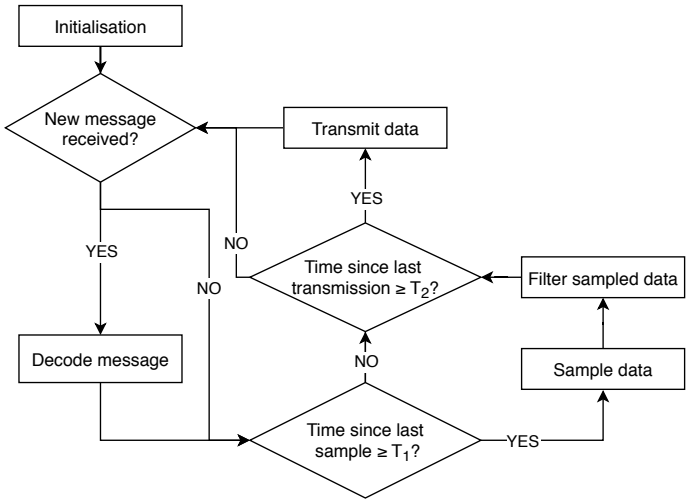


Figure 7: Flowchart of the main structure on the node. For simplicity the flowchart only illustrates logic for implementing one sensor and only shows the scenario where the sensor has received initial configuration. T_1 is the sampling period and T_2 the transmission period defined by the hub.

information to each of the sensors in order to set them up. Other than setting up the sensors, the hub will receive messages and respond if too much time passes without a message from a particular sensor. The nodes administer transmission and signal processing for the attached sensors. In order to ease reuse of code, functionalities have been attempted to be separated into library and driver files.

8 Communications

CAN is a multi-master protocol which is designed with carrier sense multiple access, *CSMA*, which means that a node cannot transmit if another node is already transmitting. The bus features a non-destructive arbitration mechanism, awarding the right to transmit on the bus to the message with highest priority. These features effectively guarantee that no messages can collide and cause errors on the bus as long as no messages are configured with identical CAN-IDs.

The signal on the bus is divided into CANH and a CANL and utilises a wired-AND function, see [21, p. 5] for an illustration of how the signals on the bus are driven. Logic 0 becomes the dominant bit, where the two voltages are actively driven apart and logic 1 becomes the recessive bit, as the signals in this state are not driven. To drive the signals on the entire bus only one node on the network needs to transmit a dominant bit, hence the dominance. The CAN controller synchronises using edges in the signals. If several 1s or 0s are transmitted in succession, edges will be far apart and synchronisation will be unreliable. Therefore CAN has implemented bit stuffing which adds a complimentary bit after 5 repeating bits. These stuff bits are destuffed at the receiving end. This functionality is also handled by the CAN controller.

8.1 Cyclic Redundancy Check

Cyclic redundancy check, CRC, is a way to help detect errors in a transmission. A CRC is carried out by first defining the data word, which is the part of the message to check for errors. The data word will be left shifted a number of times, and polynomially divided by a defined generator polynomial. The number of left shifts is equal to the degree of the generator polynomial. This division gives a remainder which will be arithmetically added to the shifted data word, yielding the code word. This code word is then transmitted, and on the receiving end, it is divided by the same generator polynomial yielding a new remainder. This remainder is referred to as the syndrome and if the syndrome equals zero, the transmission is believed to be error free. The polynomial representation is a way of depicting a bit string.

In CRC the dataword is written into polynomial form by using the bits from the dataword and multiplying them with x^n where n is the position of the bit in the dataword. For example the dataword 10101 turns into the polynomial:

$$1 \cdot x^4 + 0 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^4 + x^2 + 1 \tag{3}$$

CRC in CAN

In CAN a 15th order generator polynomial is utilised. It has the form shown in 4.

$$x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1 \tag{4}$$

In the CAN frame 15 bits are reserved for the CRC-sequence. The data word in CAN is defined by the *Start of frame*, *Arbitration Field*, *Control Field* and *Data Field*. The data word is left shifted 15 times and divided by the generator polynomial described in equation 4. The remainder of this division is placed in the CRC-section of the CAN frame. To check for errors the receiver acts as a decoder and divides the CAN message, as a code word, using the same generator polynomial, which yields a syndrome of 14 bits which must all be 0 for the CAN-message to be valid [19, p. 15-16].

Error Detection

Generally if any kind of error is divisible by the generator polynomial, if the syndrome is all 0's, the errors will not be caught. Given that the generator polynomial has more than one term and the coefficient of x^0 is 1, all single-bit errors will be caught [17, p. 271].

The CAN generator polynomial can be written as the product of two polynomials of lesser degree:

$$(x^{14} + x^9 + x^8 + x^6 + x^5 + x^4 + x^2 + x + 1)(x + 1) \tag{5}$$

Equation 5 shows that the CAN generator polynomial has the factor $(x + 1)$. Therefore it will catch all odd-numbered errors [17, p. 272].

In addition to these abilities the CAN-specification states that by incorporating all its error detection methods it will detect up to five randomly distributed errors in a message, and burst errors of length less than 15 [19, p. 10]. The cyclic redundancy check implemented in CAN does however not always meet the specifications specified by Bosch due to the stuff bits introduced in the messages. Further analysis of this and estimation of the error probability is beyond the scope of this project [34]. CAN, however, does also do error checks by validating stuff bits and frame specifications, bit monitoring, and acknowledge checks.

One of the benefits of using cyclic codes is they can be easily implemented in hardware. On the transmitting side the shifting of the data word, division with the generator polynomial, and adding of the remainder to the data word is handled in the peripherals of the micro controller. In the same way at the reception side the division of the code word by the generator polynomial and the evaluation of the syndrome is also handled by peripherals in the micro controller. So the cyclic redundancy check as a whole happens automatically. However in the digital appendix, outlined in section 14, an example of a calculation of a CRC-15 for a CAN-message is included.

8.2 Protocol

CAN dictates a frame format, but exactly how to utilise it is up to the designer of the network. This means that an additional protocol will be laid on top of the existing CAN protocol to support the functionality needed for the project. Since every sensor is regarded as an independent entity on the network, each sensor is assigned a unique CAN-ID, which is used for transmitting data related to that sensor to the hub. The hub will use the same identifier to address the sensor. This has some implications which will be discussed in section 11. It is important to note that the identifier is attached to a specific sensor, and not the node which the sensor is attached to. Assigning the CAN-IDs this way has been chosen since it is easy to comprehend. Furthermore, the CAN-IDs are divided into two sections starting 100 identifiers apart. This is done in order to allow sensors monitoring SCS to have higher priority on the bus by having a lower CAN-ID.

The CAN-ID is hard coded into the software and can not be changed while the network is running. It is up to the integrator to assign each sensor on the nodes unique CAN-IDs, when the nodes are programmed. The protocol or the system does not check if any CAN-IDs have been used multiple times.

The additional protocol, which has been developed and laid on top of CAN, defines what specific message types can be used and how they should be interpreted. All messages use the full eight bytes of payload that CAN allows. This is the case even if some bytes are not currently used and has been chosen for ease of implementation. The first byte is used to indicate the message type. There are three message types, service, error and data. The highest nibble defines the message type, while the lower nibble defines sub type if applicable.

Sub Type	Sub Type Identifier
Acknowledge from sensor	0x1
Sensor setup	0x3
Shut down sensor	0x4
Coefficient setup	0x5
Request setup information	0x8

Table 6: Sub messages for service messages.

Service message

The service message is used to pass information related to configuration and service between the hub and the sensor instances. Service messages are characterised by the type identifier 0xC and have five sub types shown in table 6, which will be briefly described below.

The coefficient setup shown in table 7c is needed to support the conversion using a polynomial described in section 7. One message for each coefficient must be sent. The messages are intended to be sent before the sensor setup message. If no coefficient setup is sent, the default configuration is used. In addition to the coefficients themselves, the message also holds an index indicating which number coefficient is being sent, and the total number of coefficients to be expected.

The sensor setup message shown in table 7b has the primary function of setting up the sensor with the desired transmission period and cutoff frequency for the digital filter. On top of this, information about the sensor is also transmitted. The cutoff frequency will be used to calculate the sampling rate of the sensor as described in section 9.2. After a setup message has been received from the hub, the sensor sends back an acknowledge message, and will begin transmitting data at the configured rate.

Byte 2 through 4 of the acknowledge message, seen on table 7a, is constructed similar to setup message. Byte 5 contains the calculated sampling frequency and byte 6 contains the total number of polynomial coefficients. This information is sent by the sensor to allow for the hub to be able to verify that the sensor has configured itself correctly based on the transmitted information. Verifying the information is not implemented, but the hub does expect to receive an acknowledge to confirm setup.

The shut down sensor and request setup information messages do not carry any additional information. The shut down sensor message is transmitted to a sensor to reset it to the default configuration where no transmission occurs. The request setup information is transmitted by a sensor to acquire setup information, if none has been received. This can occur if a node is connected while the rest of the network is already running. If the sensor is known to the hub, based on the CAN-ID, the hub can locate the correct setup configuration and transmit the appropriate coefficient setup and sensor setup messages.

byte no.	High Nibble	Low Nibble
1	0xC	0x1
2	Sensor type	Unit
3	Transmission period	
4	Cutoff frequency	
5	Sampling frequency	
6	Total number of coeffficients	
7-8	Not used	

(a)

byte no.	High Nibble	Low Nibble
1	0xC	0x3
2	Sensor type	Unit
3	Transmision	
4	Cutoff frequency	
5-8	Not used	

(b)

byte no.	High Nibble	Low Nibble
1	0xC	0x5
2	Coefficient number	Total number of Coefficients
3-6	Coefficient	
7-8	Not used	

(c)

Table 7: a) Acknowledge message from sensor, b) sensor setup message and c) Coefficient setup message.

byte no.	High Nibble	Low Nibble
1	0x3	0x0
2	Sensor type	Unit
3-6	Sensor value	
7-8	Not used	

Table 8: Data message.

byte no.	High Nibble	Low Nibble
1	0xA	0x0
2	Error type	
2-8	Not used	

Table 9: Error message.

Error Message

To allow for the sensor to report that some error has occurred, e.g. wire-off described in section 9.1, an error message type has been designed. The error message is shown in table 9 and has the type identifier 0xA. If the hub receives four error messages from one sensor, a shut down sensor message will be sent, as it is assumed that the sensor is faulty.

Data Message

The data message, shown in table 8 is the most common message sent on the network. It has the type identifier 0x3 and is used to transmit the sampled and processed data from the sensor to the hub. Along with the data, which is sent as a float taking up four bytes, the sensor type and unit is also transmitted.

8.3 Maximum Bus Load

It is wanted to estimate how many SCS and regular sensors the system can support at maximum. Thereby the integrator will have an estimation for the number of sensors that can safely be introduced to the system, without messages being delayed enough to trigger the heart beat monitor. Since the system is master/master based, no central unit can schedule transmissions. This means that due to the way CAN arbitration works, messages with low priority may be significantly delayed, if there is significant traffic on the bus. To find out how many sensors can be supported, first the maximum length of a data message including stuff bits is calculated. The theoretical maximum number of bits per message, L_{max}, can be calculated by the equation 6 [3]. The equation, in which n defines the total length of the payload, assumes a maximum number of stuff bits, which will rarely, if ever, be the case during normal operation.

$$L_{max} = 8 \cdot n + 44 + \left(\frac{34 + 8 \cdot n - 1}{5} \right) = 8 \cdot 8 + 44 + \left(\frac{34 + 8 \cdot 8 - 1}{5} \right) \approx 127\text{bits} \tag{6}$$

Next it is wanted to calculate the maximum throughput of the network and get an estimation of the maximum number of messages, N_{max}, that theoretically can be sent through the network per second. To calculate the maximum theoretical throughput of the network, equation 6 will be used together with the used bit rate, T_{rate}, of 1 Mbit/s.

$$N_{max} = \frac{T_{rate}}{L_{max}} \approx 7874 \quad \text{messages/s} \tag{7}$$

Transmission via can will not work properly if the average bus load is too high. Therefore, an average safe bus load, where no messages are delayed significantly, has to be found. Depending on the source, an estimation of this varies a lot and thus, to be safe, the lowest found will be used to estimate the number of messages, $N_{allowed}$, the system can safely handle per second. The maximum bus load used for the calculation will therefore be set to 50 % [4] equalling to 3937 messages/s.

To calculate the number of SCS sensors that can be supported, it has been determined that one half of $N_{allowed}$ may be made up of messages from SCS sensors. Furthermore it has been decided that SCS sensors will transmit at a rate of 100 Hz, yielding a maximum number of SCS sensors of 19. Regular sensors are expected to transmit at a rate of 20 Hz, and may take up the other half of $N_{allowed}$. This yields a maximum of 98 regular sensors.

8.4 Software

CAN in ATmega32M1

The ATmega32M1 was chosen because it has a built in CAN controller which provides the hardware to conveniently handle message management. Messages that are to be transmitted, or messages that have been received, are stored in so called message objects. The message objects serve as a way to describe the CAN frame as an object, meaning that it contains all the information needed to build a CAN frame. These are attributes such as CAN-ID, payload length and the payload data itself. The ATmega32M1 contains six message objects which have to be addressed using the CANPAGE register [14, p. 263].

Libraries and Drivers

As mentioned in section 4.3 it was expected that a premade library and driver set could be used to implement CAN functionality. As it turned out, it was not possible to bring it to work, and in turn a new set had to be made, heavily based on the existing. The files "CAN_lib.c" and "CAN_lib.h" contain all the functionality that makes it possible to setup and send messages via the CAN bus. The header file contains the structure "st_cmd_t". This structure attempts to mirror the message objects found in the ATmega32M1. The structure holds attributes related to the message object. These include CAN-ID, payload length and the payload data itself. It also has an attribute which allows the message object to be set up to either receive or transmit. The configuration of the message object is handled by one of the main functions in the CAN library. The library consists of three main functions: A function for initialising the CAN controller, a function used for configuring a message object based on an instance of the structure "st_cmd_t" and a function for transferring data from a message object to the structure. The function for initialising the CAN bus utilises macros and functions defined in the driver, "CAN_drv.h". The function sets the bit rate, clears the message objects and enable the CAN functionality. The transfer function is used when data needs to be moved from a message object to the structure "st_cmd_t" in order to make the data accessible for the main code.

8.5 Conclusion

The ATmega32M1 has a built in CAN controller. The CAN controller contains message objects which can be configured using functions made available by a CAN library and associated driver.

The CAN frames include message identifiers, of which each sensor on the network will be assigned one. This identifier will be used to associate messages with a specific sensor. An additional protocol has been designed to lay on top of the CAN protocol in order to accommodate the messages needed for the system. The most important message formats defined by the protocol are setup messages, messages to configure sensors, and data messages, which carry data from the sensors.

9 Signal Processing

A large part of the project involves collecting and processing data from the sensors connected to each node. Because of the electromagnetically noisy environment that the system has to operate in, it has been determined in sections 3 and 4 that some kind of digital filtering of the data sampled from the sensors will have to take place.

9.1 Sampling

The Nyquist-Shannon sampling theorem dictates that in order for a continuous-time signal to be properly sampled, so that it can be fully reconstructed, the sampling frequency must be twice that of the highest frequency component in the signal. If there exists frequency components higher than half of the sampling rate, copies of these frequencies will alias around the Nyquist-frequency preventing the signal from being able to be reconstructed [25, p. 15-20] [8]. Therefor an anti-aliasing filter is used. Typically this is a low pass filter, which is characterised by passing all frequencies in the passband and attenuating all frequencies in the stopband preventing these images from aliasing into the passband. To achieve the best performance from the filter in the form of a steeper transition between the passband and stopband, a higher order filter is needed. In order to determine the amount of attenuation needed in the stopband the ideal signal to noise ratio, SNR, is to be found. It is found by looking at the finite resolution of the ADC. As the ADC needs to quantisise and digitise the samples a quantisation error will be introduced. So if the anti-aliasing filter can attenuate unwanted frequencies below the quantisation error the unwanted frequencies will have no effect.

For an N-bit analog-to-digital converter the ideal SNR in decibel [20] is given by

$$SNR_{ADC-ideal} = 6.02\,\mathrm{dB} \cdot N + 1.76\,\mathrm{dB} = 61.96\,\mathrm{dB}. \tag{8}$$

With N equal to ten, the ideally designed anti-aliasing filter should have a 62 dB attenuation at half of the sampling frequency. If the filter had an ideal brick wall response, the cutoff frequency should be half the sampling frequency. However in reality filters are not ideal, even less so the filter used as anti-aliasing filter in this project.

For the anti-aliasing filter a passive first order RC filter is chosen. The reasons for this and the limitations and inadequacy of this filter will be discussed in section 11.4. The values of the resistor

Attenuation [dB]	Frequency [Hz]
0.1	7.6
1	25.3
20	495
62	$6.23 \cdot 10^4$

Table 10: The table shows frequency for different key attenuations.

and the capacitor are chosen to yield a cutoff frequency of 50 Hz. Since the filter is a first order filter it will provide attenuation of 20 dB/decade [1].

Table 10 shows the frequency for different levels of attenuation. The table shows that the analog filter has some attenuation in the passband. The desired attenuation of 62 dB is at a frequency of nearly 62.5 kHz or almost 1250 times the cutoff frequency.

In order for the filter to achieve a much steeper roll-off a filter with a higher order should have been utilised.

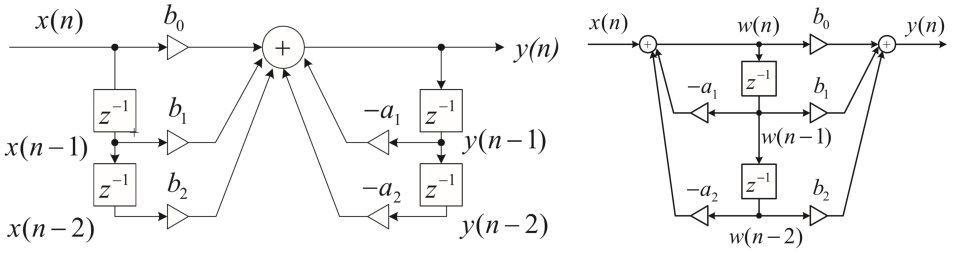
Sampling Driver

A function has been made to manage the sampling of the voltage signals from the sensors. The node has two analog sensor inputs, which will have to be converted into digital values. Since the ATmega32M1 only contains one ADC-unit, it must be possible to switch between the two inputs. Because switching between inputs occurs, it has been chosen to run only a single conversion at a time. Switching is possible by multiplexing between the pins. This is done by configuring registers specific to the micro controller [14], thus the macros for this have been written into a driver filer. When a pin has been selected and an AD-conversion has commenced, it is necessary to wait a few clock cycles for the result to be ready. The result has been chosen to be right shifted in order to utilise the full 10 bits of resolution available. The result is passed to the filter function described in section 9.2, and afterwards saved to the specific sensor structure to which it belongs.

Wire-off

A wire-off detection mechanism is implemented to ensure that if a sensor is short circuited to ground or supply voltage, the hub will be alerted. Every time a sampling has been made the filtered value is run trough a function, which transmits an error message, if the filtered value is above 4.5 V or below 0.5 V. Before the filter values are stabilised the wire-off function is at risk of being invalid, due to the fluctuation of the sampled value upon start-up. Therefore the wire-off function is only used after 100 filtered values.

By only accepting signals from 0.5 to 4.5 V the resolution of the AD-converter is reduced by 20 %, thus a lower resolution of signals than the initial 10 bit is present.



(a) Direct form I.

(b) Direct form II.

Figure 8: Implementations forms [25, p. 194-195].

9.2 Filter

Realisation of digital filters on the sensor node can be divided into two sections: A software implementation of functions that execute a given filter, and actual design of specific filters based on cutoff frequency, attenuation, etc. Since only limited amounts of data has been available regarding the signals, and associated frequency ranges, which are to be monitored, focus has been prioritised on developing a software implementation which offers a lot of customizability. This also means that less focus and thought has gone into choosing and designing the filters. The implementation is based upon an idea of having a handful of low pass filters with different cutoff frequencies, with the possibility of raising the cutoff frequency of each filter by increasing the sampling rate. That way the gaps between the cutoff frequencies of each filter can be covered and great customisation is achieved. Due to time limitations, it was ultimately decided to only design one filter to cover the range of frequencies, however the library and functions for the filters has been designed with the initial idea in mind.

Software Implementation

There exists several ways of realising a digital filter on a micro controller. The filter can be implemented using either direct-form-I or direct-form-II, which are shown in figure 8, where a and b refer to the denominator and numerator coefficients of the transfer function of the filter. Each of the boxes containing z^{-1} indicates a value from a previous sample which has to be stored. Based on this, the direct-form-II is preferable for this application as it will require less values to be stored and loaded, thereby saving execution time and memory space.

If the filter to be implemented is of high order, it is often split into second order sections that are either summed, as seen in equation 9 or multiplied, as seen in equation 10 make up the original filter. Doing this will reduce the error that can arise from numerical inaccuracies that exists because the numbers in the processor only have a finite number of bits to be stored in. A realisation structure where the sections are summed is referred to as parallel, as it allows each sections to be calculated individually. The multiplication structure is referred to as a cascade or series implementation as the output of each section will be fed as input to the next section. The latter is chosen for this application since using the parallel will not be particularly advantageous as only one processing

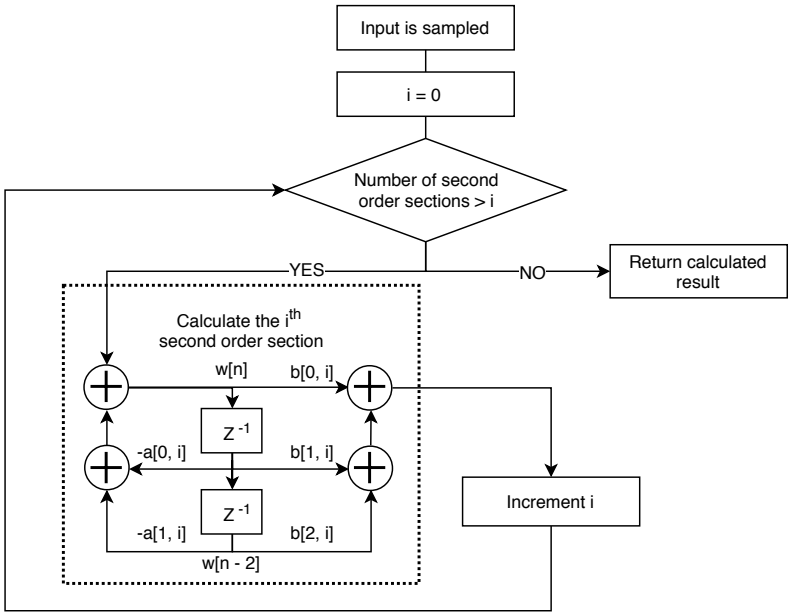


Figure 9: Flowchart describing the execution of the filter function.

core is available. Ultimately, the implementation of the filter function follows the structure of figure 9.

$$H(z) = H_1(z) + H_2(z) + H_3(z) + ... + H_n(z) \tag{9}$$

$$H(z) = H_1(z) \cdot H_2(z) \cdot H_3(z) \cdot ... \cdot H_n(z) \tag{10}$$

In order to easily allow for several different filters for the system, a structure is defined to hold values needed to describe the filter. The structure holds the filter order and two two-dimensional arrays for the numerator and denominator coefficients respectively. On top of this, it also holds a value that indicates the cutoff frequency at a sampling frequency of 100 Hz. This value will be used to select the correct sampling rate in order to achieve the desired cutoff frequency. Based on the desired cutoff frequency, f_{cd} and the cutoff frequency at the sampling rate of 100 Hz, f_{c100}, the required sampling frequency can be calculated as described in equation 11.

$$f_s = \frac{f_{cd}}{f_{c100}} \cdot 100\,\text{Hz} \tag{11}$$

Sampling timed based on intervals of 0.5 ms. This means that the period between samplings must be truncated to a precision of 0.5 ms resulting in a finite number of available cutoff frequencies. It also cannot be guaranteed that the exact desired cutoff frequency is achieved. Another slight disadvantage of changing the cutoff frequency by altering the sampling rate is that by raising the sampling rate, and thereby the cutoff frequency, the transition period of the filter is also prolonged.

The filter, which follows the logic described in figure 9, has been implemented as a set of functions. One function that calculates a second order section and a main function which repeatedly calls the

first function and feeds the results back into it as it iterates through each second-order section. As parameters the main function takes an input value, a filter structure, which has previously been described, and an array of structures that act as buffers. The buffers will hold $w[n-i]$ values from previous runs of the filter. Therefore, the required number of buffers will be equal to the number of second order sections necessary to compute the filter. The buffers have been designed as a set consisting of an array and a pointer used for indexing the array. This has been chosen in order to establish a circular buffer. This is beneficial as it eliminates the need to rearrange the entire buffer each time a new value of w is added.

The buffers are kept separate from the filter structure in order to allow both sensors to reference the same filter. This is why the array of buffers are part of the sensor structure, whereas the filter is only pointed to by a pointer in the sensor structure. This means that the buffers which contain the previous w values are "private" to each sensor whereas the filter definition can be shared.

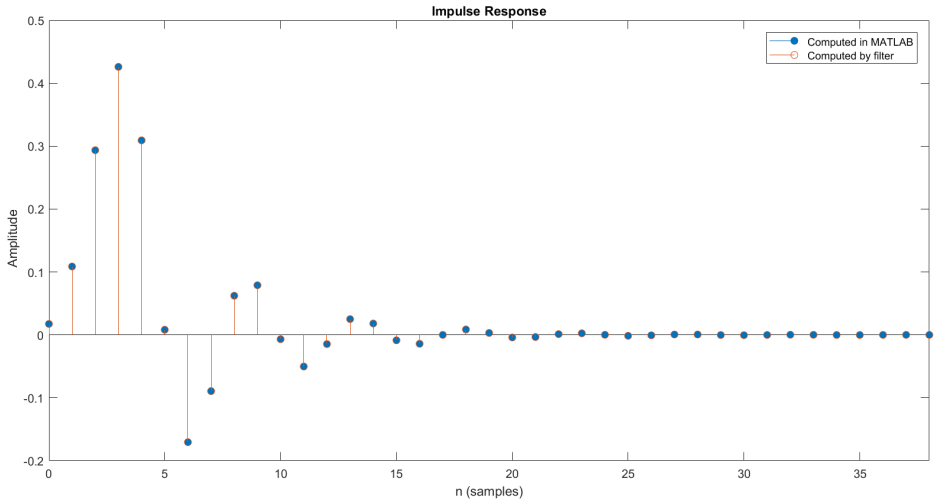


Figure 10: Impulse Response of the tested filter compared with a MATLAB computed result.

To verify the implementation of the filter function, a test filter is made. The impulse response of the test filter is compared with an impulse response found using the same values in MATLAB, shown in figure 10. The average deviation between the two has been calculated to be $5{,}22 \cdot 10^{-4}\%$.

Designing the Filter

When designing a filter, a range of choices must be made. These include features such as cutoff frequency, transition period and stopband attenuation. As mentioned previously, selecting the ideal cutoff frequency has not been a great focus for this project. However, for the filter that has been designed quite some consideration has gone into choosing the correct filter type. Since the digital filter has to be implemented and executed on the relatively slow ATmega32M1 processor, which is clocked at 16 MHz, it is also important to factor in the order of the filter. A higher order filter will result in a steeper transition between passband and stopband, however it will also take

longer time to execute, as it requires more calculations to be processed. In section 4 it has been determined that execution time of the filter must not exceed 500 µs. This is important to consider when choosing which filter to use and how to implement it. A feature which is worth deciding upon at this stage is whether to use a finite impulse response, FIR, filter or an infinite impulse response, IIR, filter. FIR filters depend only on current and previous inputs, whereas an IIR filter depends on both the current input as well as previous inputs and outputs. Table 11 highlights some of the general pros and cons of each type.

Solution	**Advantages**	**Disadvantages**
FIR	• Easy to design for linear phase. • Always stable since the denominator of the transfer function is always 1.	• Requires high filter order to achieve short transition with high attenuation.
IIR	• High attenuation and short transition can be achieved with a relatively low filter order.	• Non-linear phase. • The filters can become unstable.

Table 11: Advantages and disadvantages of FIR and IIR filters.

It is chosen to implement an IIR filter since having a lower filter order will be vital to keeping the computation time down. The non-linear phase is not an issue for this application, however when designing the filter it must be ensured that it is stable.

The filter has been designed using the bilinear transformation method, using a Chebyshev type II filter as the analog prototype filter. The Chebyshev type II filter has been chosen because it has a relatively steep transition period while conserving the passband free of ripple. Deciding exactly where to place the cutoff frequency for the filter is difficult since the signals, and therefore their frequency ranges, are not well defined. This is why, during the project less focus has been put on picking out and designing the perfect filters, and instead focus has been on implementing the filter in a way that offers customizability. Therefore, a MATLAB script, which can be found in the digital appendix outlined in section 14, has been developed. Based on user inputs it will output filter coefficients to be used with the implemented filter function.

In section 3 some estimates have been presented, and choice of cutoff frequency is based on these. It has been determined that the signals to be monitored have no interesting frequency components higher than 10 Hz. Therefore a base cutoff frequency of 10 Hz has been used for the filter implemented on the node.

When designing an IIR filter it is imperative to know if the filter is stable, thus a pole-zero plot, shown on figure 12, has been made. The poles all lie within the unit-circle indicating that the filter is stable. Figure 13 shows the frequency response of the designed low pass filter, with a cutoff frequency of approximately 10 Hz and a stopband frequency at approximately 23 Hz. The characteristic ripple in the stopband for a Chebyshev type II is also visible.

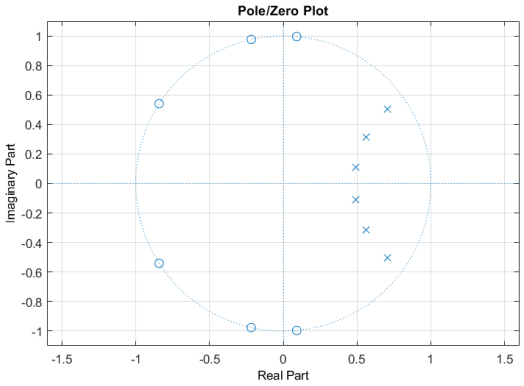


Table 12: Pole-zero plot of the filter.

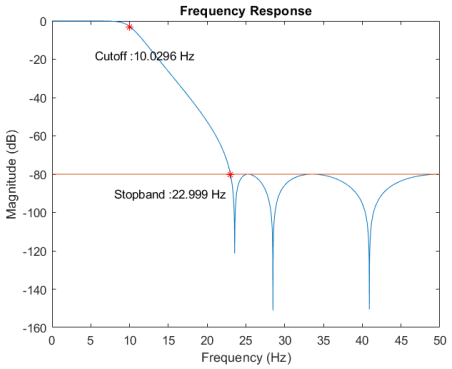


Table 13: Frequency response of the filter with cutoff and stopband frequencies marked with a red dot.

9.3 Conclusion

To eliminate aliases in the sampled signal a first order anti-aliasing RC filter, with a cutoff frequency of 50 Hz is implemented. It was a poor decision and is discussed in section 11. The transition period is not seen as ideal and is further discussed in section 11. The sensor inputs are sampled by converting the analog signal to a digital value. This value is then filtered using a digital filter realised in software using cascades of second order direct form II sections. Using MATLAB it has been verified to work as intended. Utilising the short transition period a digital Chebyshev type II filter the data is filtered with a base cutoff frequency of 10 Hz. The cutoff frequency can be adjusted by altering the sampling frequency. To signify any errors with the sensor a wire-off function is implemented to alert the hub if the filtered data values are above 4.5 V or below 0.5 V.

10 Tests

This section concerns the testing the requirements of the network specified in section 5. Procedures and results of the tests are elaborated further in the test journals [11], which can be found in the digital appendix. The logs and other raw test data can also be found in the digital appendix.

10.1 Test Network

A test network have been defined for the purpose to assure that all test are conducted with some degree of cohesion. The defined network is configured as seen in table 14, unless stated otherwise. A visualisation of the test network can be seen on figure 11. All sensor inputs are simulated by the use of potentiometers. A *CAN logger* is used to monitor the network during testing.

No.	ID	SCS	Unit	Sensor Type	Polynomial	Transmission Period	Cutoff Frequency
1	200	0	Other	Potentiometer	x	10 s	10 Hz
2	202	0	%	Potentiometer	x	10 s	12 Hz
3	100	1	%	Potentiometer	-12.5+25x	5 s	15 Hz
4	204	0	%	Potentiometer	x	10 s	20 Hz
5	206	0	°C	Thermistor	$2x + x^2$	10 s	10 Hz

Table 14: List of sensor and their configurations on the network. Structure of the network can be seen on figure 11.

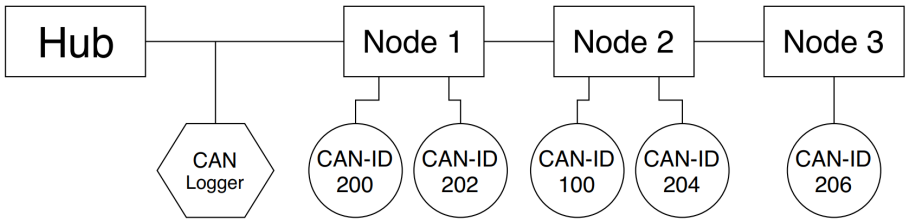


Figure 11: A schematic of the network configuration of the test network consisting of one hub, three nodes, five sensors with displayed CAN-IDs and a CAN logger to monitor the network.

10.2 Power-off Test

Powering a system off and on repeatedly is a great indication of the robustness of the system. The requirements tested are:

- The network system must be able to start up successfully within 100 ms. Successful start-up is confirmed when all entities on the network have verified their initialisation.
- Each node is to be able to request setup instructions upon connecting to the network.

As the requirements indicate, the test is divided into two parts, one test for a single node and one for the whole network. Initialisation of sensor entities on the network is confirmed by sending acknowledge messages. A node is deemed operational when data messages have been transmitted from the related sensors. Thus the time from start-up to these specific events are measured.

With these results it is to be noted, that there is a 100 ms delay start-up, when a single node is set up. The delay is present to prevent the nodes to request setup upon start-up. Thus the relative long period for a single node seen in table 16. The delay is reflected in the results as the sensor entities acknowledge their configuration, significantly earlier than they begin transmitting data messages.

Five measurements are made giving a start-up time and showing that the network can be consistently powered off and on. This is true for the network as well as a single node. As shown in

Test	Time [ms]
1-4	22.88
5	22.92
Average	22.89

Table 15: Complete network power-off test.

Test	Time [ms]
1-4	109.8
5	102.2
Average	108.3

Table 16: Single node power-off test.

table 15 the network is able to start up in 22.89 ms on average satisfying the initial requirement of 100 ms with the given test configurations. The average time of start-up will increase as more nodes and sensors are added, since more transmissions will be conducted.

10.3 Stress Test

The following requirements from section 5 is to be tested under a high bus load.

- A maximum bus load at which the operation of the system is guaranteed must be determined. At this load 1000/1000 messages must be received by the hub, and no messages may be postponed by more than what is equal to transmission of two messages.

In section 8.3, the maximum bus load for guaranteed operation is determined to be 50 %. The transmission time of a single message is found as:

$$t_{msg} = \frac{msg_{size}}{P} = \frac{127 \text{ bit}}{1 \text{ Mbps}} = 127 \,\mu\text{s} \quad , \tag{12}$$

where msg_{size} is the size of a message and P is the chosen bit rate of CAN.

After initialisation, a fixed number of messages are sent by each sensor on the test network. The number of messages as well as transmission rate configuration for the sensors are varied between five tests. At the end of transmission, the hub transmits the total number of messages it has received. As it is shown in table 17 even at a bus load much higher than the determined maximum, 9007/9007 messages have been received by the hub. The seemingly extra seven messages are due to initial setup. The whole log of messages is not available for test four and five, and therefore it is not possible with certainty to verify that seven setup messages exist for these two tests, however it is expected. It is evident from this test that reception of messages on the hub, even at high bus loads is not an issue.

Test number two has been selected for analysis of transmission continuity since it has the closest bus load to the chosen maximum. Figure 12 shows that four out of five of the sensors transmit without being suppressed, but the lowest priority sensor has been suppressed for a brief period. From figure 12b it is evident that some form of transmission burst has happened immediately after setup in which the sensors do not follow their intended transmission frequency. This is caused by an error in the program for the sensor node that has been fixed. The figure also shows, that at the point when all the sensors conform to their intended transmission rate, messages are distributed evenly.

Test no.	Transmission Rate [kHz]					Msg. per Sensor	Total Msg. Transmitted	Total Msg. Received
1	2	2	2	2	2	1800	9007	9007
2	1.5	1.5	1.5	1.5	1.5	1800	9007	9007
3	0.2	0.2	0.4	0.4	0.4	1800	9007	9007
4	2	2	2	2	2	3000	-	15007
5	2	2	2	2	2	10000	-	50007

Table 17: Results from stress test. The configuration column describes the transmission rate for each of the five sensors.

It has been attempted to examine whether messages are delayed calculating the time period between each of the 1800 messages transmitted by the sensor with CAN-ID 206. Due to the time stamps from the CAN logger being limited to 1 ms of precision, it is not possible to give an exact indication of the deviation in the transmission period. With the limitations from the CAN logger it is not possible to guarantee that the time between messages is below 0.254 ms, which is discussed in section 11.

10.4 Wire-off Detection

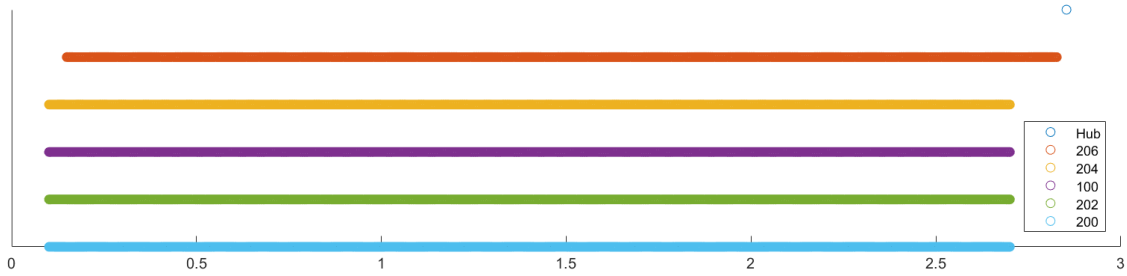
The following requirement is tested:

- Each node must be able to detect so called *wire-off* errors, if the input from a sensor goes below 0.5 V or above 4.5 V.

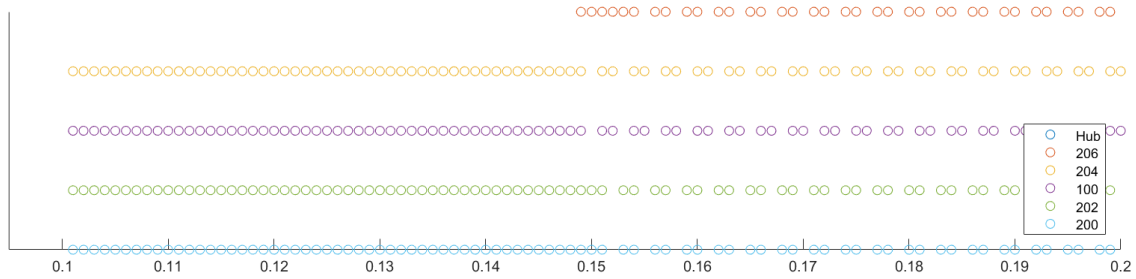
Furthermore it is wanted to test, functionality which ensures the hub will shut down a sensor, if a total of 4 error messages has been sent.

A test is designed to examine if the wire-off functionality will enable after 100 samples as described in section 9.1. The test was performed by connecting a potentiometer to a node connected to the bus. The potentiometer was initially turned to max, in order to ensure the filter output would exceed 4.5 V as fast as possible. The filter output was transmitted on the bus after every sampling. A total of 4 error messages are sent from a sensor, before the sensor is shut down by the hub. By analysing the data, which can be found in the journal [11, p. 16], it can be observed that the first error message is sent after exactly 100 samples, and the sensor is shut down after 4 error messages.

A second test will examine if the wire-off functionality triggers at the expected thresholds. From a central position, the potentiometer is turned to output a voltage lower than the minimum threshold. On figure 13 the data from the test is plotted. The sensor is shut down after 4 error messages as expected, however if the plot is examined closer its possible to see that the wire-off function has not been triggered at the desired threshold, instead the first error message has been sent at approximately 0.7 V instead of 0.5 V. In all probability this is happening due to a discovered error,



(a) Stress test number two. Circles are overlapping creating a line. The one message transmitted from the hub, is the message which holds the information regarding the number of received messages.



(b) Stress test number two zoomed in at the very beginning.

Figure 12: Transmission flow for test number two. Each circle represents a message.

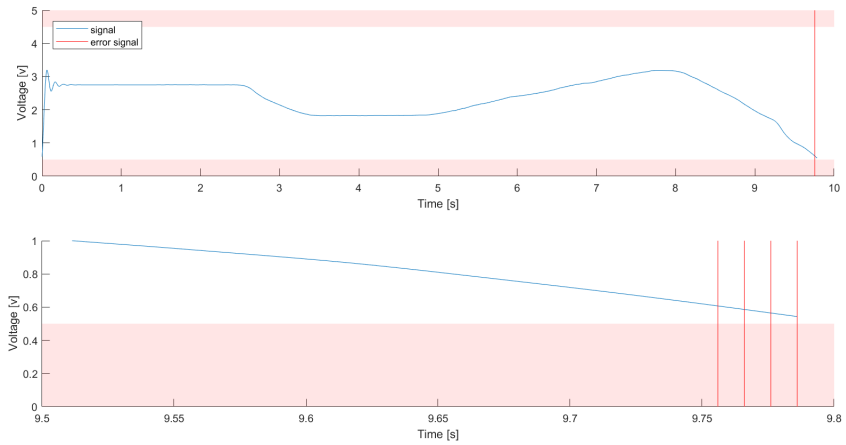


Figure 13: Plot of filter output and error messages.

	With Wire-off	**Without Wire-off**
Average Time	450.57 μs	306.08 μs

Table 18: Duration of the sample and filter code implemented on the sensor.

where the wire-off function is monitoring the raw sensor input, and not on the filtered data as intended. The wire-off function might be triggering at the right threshold, but due to the fact, that the filtered data is a bit delayed compared to the raw sensor input, it cannot be verified. The error has been rectified in the final program.

A third test has been conducted to ensure that the functionality works in a system with multiple sensors. The test, described in the journal [11, p. 19], showed the system working as expected based on what has been shown in the previous tests involving a single sensor.

10.5 Execution Time for Sampling and Filtering

The following requirement regarding sampling and filtering is tested.

- At a sampling frequency of 500 Hz, execution of digital filters for two sensor inputs may use no more than 50 % of the CPU time.

A test is conducted with and without wire-off being triggered, since having to transmit an error message prolongs the execution time. The requirement dictates, that the time it takes to sample and filter data is not to exceed 0.5 ms.

The results of the test, shown in table 18, show that with a maximum execution time of 450.57 µs, the requirement is met.

Figure 14 shows an estimation of the work load distribution on the node. Based on the execution times found in table 18, the CPU uses about 45 % on sampling and filtering. By subtracting the results for the execution times with and without transmission of error messages an estimate of general transmission time is achieved. By assuming a maximum average transmission frequency for a sensor of 100 Hz, 3 % of the CPU time will be spent on transmission with two sensors per node. Thus a total of 52 % of the CPU time is available for other tasks. The calculation of the converting polynomial has not been included in the transmission time. It is estimated that function, at worst, will double the transmission time, leaving 49 % CPU time available.

10.6 Heart Beat Monitor

The following requirement is set for this function:

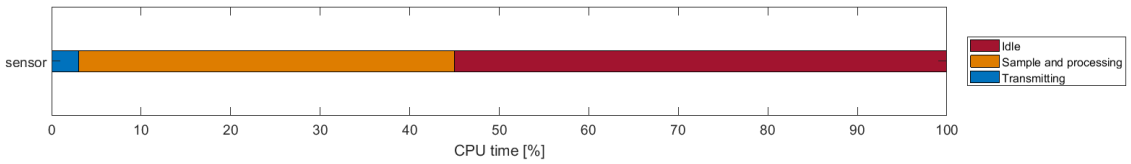


Figure 14: CPU time distribution on the sensor node.

- The hub must give an alert, within 50 ms, if transmission of messages related to an SCS is halted.

Furthermore it has been implemented that the heart beat monitor also applies to regular sensors, as described in section 7.2, requiring a response time of maximum 250 ms.

The test is conducted by initialising the test network and after a few seconds transmission is cancelled for the SCS sensor with CAN-ID 100, and later for the regular sensor with CAN-ID 200. The response time, i.e. the time between the last message is received from the particular sensor until the hub gives an alert, is tracked. The results, shown in Table 19, show that all transmission halts from SCS sensors were responded to within 50 ms and for regular sensors within 250 ms. Figure 15 shows all the messages during test number three. It is evident that directly following the halt of the message stream of a sensor, the hub sends an alert. The rest of the network continues operation as normal. While the test shows a compliance with the requirement, a slight error has been discovered. Test one shows a response time for the regular sensor of 45 ms. The way the heart beat monitor is intended to work, this should not be possible. As it turned out, this is caused by an error where the counter, which determines if new messages have been received during the last period, is reset for every sensor each time the SCS sensors are checked. This means that the counter for regular sensors is cleared five times as often as it is checked on. This leads to a risk of false alarms if regular signals transmit with more than 25 ms between messages. The error has been rectified in the final program.

	Test No.	Last Message	Alert Message	Resp. Time [ms]
SCS Signal CAN-ID 100	1	00 : 04 : 01.856	00 : 04 : 01.901	45
	2	00 : 00 : 06.189	00 : 00 : 06.224	35
	3	00 : 00 : 03.994	00 : 00 : 04.036	42
Regular Signal CAN-ID 200	1	00 : 04 : 03.335	00 : 04 : 03.380	45
	2	00 : 00 : 07.704	00 : 00 : 07.855	151
	3	00 : 00 : 05.319	00 : 00 : 05.463	144

Table 19: Results from the heart beat monitor test.

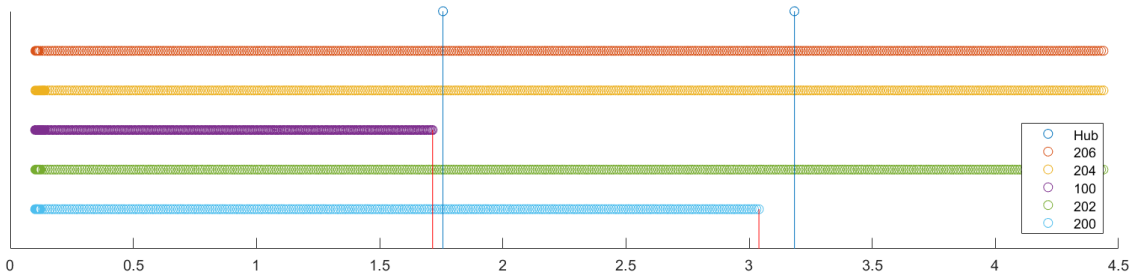


Figure 15: Plot of messages transmitted on the network during heart beat monitor test 3. Each circle represents a message. The red line indicates the last message transmitted by a sensor. The blue line represents the response by the hub.

11 Discussion

11.1 Stress Test

The stress test described in section 10 tries to expose flaws in the network when the network is under a lot of stress. To achieve this, the bus load must be high. Only by configuring the nodes with highly unrealistic values, the desired bus load could be achieved. So in order to adequately and realistically test the network a much bigger network is be needed. Also troublesome was the CAN logger used for the project. The resolution on the time stamps is in milliseconds and since transmission frequencies in the test network could be as high as 2 kHz the time stamps would need a resolution of 10 µs. The CAN logger was only able to log up to 10.000 messages, meaning that large tests have not been able to be logged.

To have a test reflecting a real scenario, it would be advantageous to have a much larger network, thus be able to stress the network without having unnatural transmission rates. With lower transmission rates the resolution of the CAN logger could be sufficient, although it is still desired to be able to log more data.

11.2 Assigning CAN-ID

As described in section 8.2 each sensor on the network is assigned a specific CAN-ID, which is used to address and identify the sensor. This entails that each sensor will always transmit using the same CAN-ID which will also be used by the hub to transmit messages to that sensor. This was chosen since it is easy to comprehend and simple to implement. However, it has become apparent that doing this leads to a risk of collisions between messages with the same CAN-ID. CAN is designed with an arbitration system which is supposed to avoid collisions, but allowing multiple entities to transmit using the same CAN-ID defeats this purpose. Under normal operation it should not be an issue since the hub is not supposed to transmit after the initial setup, however it can happen and therefore a different approach to the distribution of CAN-IDs have been thought of.

Instead of using the first byte of the payload to determine the message type, as described in section

8, the message type can be embedded in the CAN-ID. The CAN frame provides 11 bits for the CAN-ID. If the four most significant bits are used to determine the message type, that will allow for 16 different message types, leaving the remaining seven bits to identify the sensor. This allows for 128 sensors on the network. By assigning the CAN-IDs this way, no two entities on the network will ever transmit using the same CAN-ID. This is ensured because the hub transmits different message types to the sensors, and the last seven bits will be unique for each sensor. There are some great advantages to using this solution. The most apparent one is the fact that message collisions can be completely avoided, which is how CAN is intended to be used. On top of this, it will be possible to assign priority to certain message types over others. That way it would be possible to prioritise SCS messages without having to change the ID which is specific to the sensor. At last, it would also free up one more byte in the payload.

11.3 Future Software Functionality

At the moment only one digital filter is available on the system, a 6th order Chebyshev type II. The filter coefficients are stored in the program memory. For extra functionality multiple filters could be added to the system. The integrator would then have the option of choosing between different filter types like Bessel, Chebyshevs, Butterworth and Elliptic in different configurations like bandstop, bandpass, low pass high pass to suit a different sensor like characteristics or to choose to not use any filter at all.

More Demanding Sensors

At the moment the network only supports sensors with analog outputs. To further extend the capabilities of each node they could be configured to support sensors that have outputs in the form of varying duty cycles, varying signal frequencies or through SPI. Both pins routed from the micro controller to the sensor inputs support external interrupts, while one of the pins also supports input capture to one of the built-in timers in the micro controller. Therefore by configuring the software for the node it will be able to measure duty cycle and signal frequency, thereby increasing the range of sensors that can be interfaced to. The ATmega32M1 also has SPI, though it is not routed from the micro controller, so in order to interface with sensors through SPI, the node will have to be redesigned.

Porting to the Viking X Main Controller

The hub code being the main code as well as the different libraries and drivers are meant to be ported to the main controller of the Viking X, thus making the hub code a task which the controller can schedule as needed. In order for an integrator to set up the task to run properly, additional functions must be developed. These include get-, set- and add-functions that will allow the integrator to define and configure the network layout on the main controller. On top of this, the heart beat monitor must also be augmented to include a call back function. This will allow the integrator to define how the main controller shall respond to heart beat errors.

It is tried to avoid building the libraries hardware specific and instead utilise the drivers. This is acknowledged to be a very difficult discipline, which has not been achieved to the fullest in the project. Originally CAN drivers were not expected to be developed, with multiple different libraries and drivers available on the internet. Early in the process it was recognised, that the complexity of understanding the libraries and drivers to a point, where it was possible to build functions depending on them, was too great, resulting in the development of the used drivers.

Boot Loader

At the moment the node is reprogrammed through the SPI. However this interface is inaccessible when the casing is fitted to the node, and although it is easy to unscrew the casing, it is preferable to find a way to program the node without removing the casing. This is possible through the CAN interface and through the Atmel software utility called FLIP. However this requires a boot loader.

The ATmega32M1 contains 32 Kbytes of On-chip In-System Reprogrammable Flash memory. At the bottom of this memory there is 2 Kbytes of boot loader section which can be configured to not being overwritten, whenever the flash memory is reprogrammed. In this section a program called a boot loader can be placed. This boot loader can reprogram the flash through messages received on the CAN bus.

A boot loader with this functionality is available for Atmels deprecated AT90CAN devices, so it would need to be ported to the ATmega32M1. There is also a concern that the boot loader size is bigger than the available space in the boot loader section. This can be solved by switching the micro controller used to another in the family the ATmega64M1, which has doubled the flash memory and boot loader section compared to the ATmega32M1. Since this micro controller is in the same family of micro controllers as the ATmega32M1 peripherals and setting are identical and no changes to the software are needed.

By hard coding CAN-IDs in the EEPROM or the boot loader section of the micro controllers it might even be possible to flash individual micro controllers while still installed in the network.

11.4 Anti-aliasing Filter

The anti-aliasing filter used in the project is a first order RC filter. As presented in 9.1 the filter achieves the required attenuation at approximately 1250 times the cutoff frequency. This is far from ideal. For example to meet the requirements of a cutoff frequency of 50 Hz, the necessary attenuation is achieved at 62.5 kHz, which requires that the sampling frequency must be twice this to adequately prevent aliasing. Another way of achieving the required attenuation at 50 Hz the cutoff frequency would have to be set at 0.04 Hz, which is not feasible for the sensor that are to be interfaced with. Some possible solutions have been identified and are described in this section.

Maximising Sample Rate

The ADC is able to sample with a frequency of up to 125 kHz. This sampling rate is just enough to satisfy the Nyquist-Shannon sampling theorem. However this sampling frequency requires that only one ADC-channel is used, limiting the number of sensors on the node to one. A sampling frequency of 125 kHz leaves 8 µs between samples. With a clock speed of 16 MHz this yields a total of 128 clock cycles between samples. This means that the execution of the digital filter including interrupts that might trigger within this period must execute in less than 128 clock cycles. With the current digital filter setup and functionality in the interrupts this approach is simply not realistic.

Change Hardware

Without altering the dimensions of the casing there is room for a bigger PCB, and on the PCB itself there is room for more components. The extra space on the PCB could be used for deploying bigger analog filters. 8th-order analog Bessel/Butterworth/Elliptic filters are commercially available in 8-pin SO-packages. Table 20 shows differences between the types of filters.

Ultimately MAX7404 is chosen as the analog filter to analyse. Although an elliptic filter has ripple in the passband, according to the datasheet the ripple of the elliptic filter typically is between -0.11 and 0.14 dB, which is tolerable considering the benefits of the filter [30, p. 5]. The attenuation falls a little short of the required 62 dB of attenuation, but its is very close to, which is shown in calculation 13 where a voltage spike of 5 V is assumed and calculated with an attenuation of 60 dB:

$$V_{out} = 10^{\left(\frac{gain}{20}\right)} \cdot V_{in} = 10^{\left(\frac{-60}{20}\right)} \cdot 5 = 5\,\text{mV} \tag{13}$$

which is just above the quantisation level of 4.88 mV. In conclusion a 5 V spike aliasing into the passband will in the worst case have an amplitude of 5 mV that in the worst case would cause the input read at the ADC to be 2 levels higher than without the error. This gives an absolute maximum error of 0.19 %.

In the digital appendix in the electronics folder outlined by the overview in section 14, a schematic is provided, that outlines one possible implementation of the active analog filter given the available pins on the micro controller. An available pin from the micro controller is used to activate/deactivate the active filter. Since the switched capacitor filter, *SCF*, is driven by a clock, frequencies around the clock frequency of the filter will alias into the passband. Therefore there must be an RC filter before the input of the SCF with a cutoff frequency above that of the SCF cutoff frequency. At the output there is a low pass filter to attenuate the clock feedthrough from the component [30].

11.5 Digital Filter

It has become apparent that choosing and designing an anti-aliasing filter is not a trivial task. This has lead to the verdict that the approach, which was chosen to accommodate the requirement

Component	Filter Type	Attenuation [dB]	Pros	Cons
MAX7401	Bessel	33 ($3f_c$)	• Good for multiplexing • No overshoot or ringing	• Widest transition band. • Attenuation in passband.
MAX7400	Elliptic 1.5	82 ($1.5f_c$)	• Steep roll-off. • Enough attenuation.	• Ripple in passband. • Overshoot and ringing. • Poor transient response.
MAX7404	Elliptic 1.2	60 ($1.2f_c$)	• Steepest roll-off.	• Ripple in passband. • Falls just short of having enough attenuation. • Poor transient response
MAX7480	Butterworth	48 ($2f_c$)	• Flattest passband.	• Wide transition band. • Poor transient response.

Table 20: Comparison between different analog filters. All filters are 8th order switched capacitor filters, SCF, and an adjustable cutoff frequency driven by an external or internal clock. The clock to cutoff frequency-ratio is 100:1. For each of filter types the stated attenuation is achieved at the frequency expressed in the parentheses, where f_c is the chosen cutoff frequency [30] [31] [32].

of different cutoff frequencies for the digital filter, is not ideal. This approach heavily relies on changing the sampling rate, which changes the requirements for the anti-aliasing filter. When having to support sampling frequencies between 100 Hz and 500 Hz, the anti-aliasing filter, which has a fixed cut off frequency, must be designed to support the lowest sampling frequency. So unless a very high order anti-aliasing filter is used, due to the transition band, the dynamic range of the signals that can be supported will be significantly reduced, which ruins the point of having customizable filters. An alternative, and in hindsight probably more favourable, approach would be to select a fairly high fixed sampling rate. This makes selecting the anti-aliasing filter much simpler. It would require several different filter configurations in order to cover a range of cut off frequencies, however this would also make each filter perform better in regards to transition time.

11.6 The receive interrupt

As examined in the time test in section 10, the receive interrupt takes a great amount of time to execute in the worst case scenario with approximately 4.2 ms. Good coding practice is to only set flags, which the main code can react upon, thus not perform any logic in the interrupt and minimise the time spent in the interrupt. The way the receive interrupt is implemented on the hub does not complies very poorly with this. In contrast, the receive interrupt on the node is build in a more preferable way, with close to no logic being processed in the interrupt. A similar design could optimise the hub substantially, and would be considered better coding practice.

Another hazard dealing with variables in interrupts is overwriting data which is being processed in the main code. The interrupt shares variables with the main code resulting in a race condition. Race conditions can be fixed by simply disabling the conflicting interrupt before calling the function, thus ensuring that the variable will not change. A potential race condition is present on the sensor node, where a messages is loaded into a receive mob structure, thus enabling the main code to process the data. While the sensor processes the newly received message, it is possible to receive a new message, thus introducing the possibility of ambiguity. While the hazard is present on the sensor it is not likely to happen very often, due to it only receiving instructions, which will not occur during normal operation. Race condition is less likely to happen on the hub, due to all the logic being placed in the receive interrupt.

12 Conclusion

A network of interconnected nodes has been developed in order to monitor a series of sensors and transmit the sampled data to a central hub. Each node is designed around an ATmega32M1 and can interface two sensors, however the network is structured in a way where the hub perceives each sensor as an individual entity. To protect the nodes from water and dirt, a protective casing has been designed and 3D-printed. The nodes are connected using a shielded bus cable which holds two sets of twisted wire pairs to accommodate the differential signals utilised by the CAN communications protocol as well as voltage supply and ground. When the network is powered on, the hub will send messages to the network on how each of the sensor entities shall be configured.

Through tests where the network is continuously turned off and on, it has been observed that all sensors are consistently set up in less than 100 ms. In a stress test where the network is running at a bus load close to 50 %, which has been deemed to be the maximum load where operation is guaranteed, the hub has consistently shown to receive 9007/9007 messages transmitted to it. Due to limitations in testing equipment it cannot be confirmed that no messages are postponed more than two message lengths. Additional to the hub being able to configure all sensors at start-up, it has also been implemented that sensor entities related to a node which is connected to the network post start-up will request to be configured. Through tests, sensor entities connected after initial start-up have shown to consistently be correctly configured.

To allow messages to be transmitted, received and interpreted correctly in the system, a protocol has been designed to be layered on top of CAN. The protocol defines that each sensor entity is given one unique CAN-ID which is used for both reception and transmission. Complications about this approach are discussed. Furthermore a range of different message types are defined such as setup, data and error messages. An error message is transmitted by a sensor entity when it detects that the sampled signal has gone above 4.5 V or below 0.5 V. During tests error messages have been observed to consistently be transmitted when attempts were made at triggering them. However, due to an erroneous implementation, execution may not be as expected when transferred to a noisy environment.

To be able to detect whether a sensor has halted transmission, a heart beat monitor is implemented on the hub to give an alert if no SCS messages are received within 50 ms. It is chosen to also include regular signals however with a response time of 250 ms. The function is tested and is shown to work as intended for SCS messages.

A digital filter is realised using a cascade of second order direct form II structures, and implemented on the node. Using filter coefficients based on a sixths order Chebyshev type II low pass filter, it is intended for high frequency noise to be filtered out. The performance of the filter has been verified by comparing the impulse response to one produced using MATLAB. Different cutoff frequencies can be used by varying the sample rate. To avoid aliases when sampling, a first order RC filter is used for anti-aliasing. This has been discovered to be a poor choice and the consequences have been discussed. The processing time of sampling, filtering and transmitting data has been measured to estimate the workload of the micro processor. Executing sampling and filtering for two sensors has been measured to take up 45 % of the CPU time at most. Furthermore, including transmission around 50 % of the processing time is used.

Before transmission, the sampled data is converted from a voltage to the desired format, e.g. temperature, using a polynomial describing the relation between the two. The function has not been formally tested, and thus cannot be guaranteed to work, but from observation it seems to function as intended.

Ultimately a system has been developed which, when ported, is ready to be integrated in the Viking X racing car.

13 References

[1] Passive Low Pass Filter. https://www.electronics-tutorials.ws/filter/filter_2.html. Accessed: 2018-12-02.

[2] Basics of Embedded C Program. https://www.electronicshub.org/basics-of-embedded-c-program/, oct 2017. Accessed: 2018-12-11.

[3] CAN bus. https://en.wikipedia.org/wiki/CAN_bus, dec 2018. Accessed: 2018-12-12.

[4] Designing a CAN network. https://www.can-cia.org/can-knowledge/can/design-can-network/, dec 2018.

[5] Differential Signals. https://www.allaboutcircuits.com/technical-articles/the-why-and-how-of-differential-signaling/, dec 2018. Accessed: 2018-12-11.

[6] I^2C. https://en.wikipedia.org/wiki/I2C, oct 2018. Accessed: 2018-12-03.

[7] ModBus. https://www.automation.com/library/articles-white-papers/fieldbus-serial-bus-io-networks/introduction-to-modbus, dec 2018. Accessed: 2018-12-11.

[8] Nyquist-Shannon sampling theorem. https://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem, dec 2018. Accessed: 2018-12-12.

[9] RS-232. https://en.wikipedia.org/wiki/RS-232, dec 2018. Accessed: 2018-12-04.

[10] Serial Peripheral Interface. https://en.wikipedia.org/wiki/Serial_Peripheral_Interface, nov 2018. Accessed: 2018-12-03.

[11] G. 01. *3rd Semester Project PRO3 Robotics Test Journals*, dec 2018. Consists of all test journals of PRO3.

[12] AKER. *S7 Series Clock Oscillator*, april 2006.

[13] AMK. *Racing kit 4 wheel drive, Formula Student Electric*, 33 edition, 2017. Inverters used on the Viking X.

[14] A. Corporation. *ATmega16M1/32M1/64M1*, 8209f edition, oct 2016.

[15] C. Electronics. CAN Bus Explained - A Simple Intro (2018). https://www.csselectronics.com/screen/page/simple-intro-to-can-bus/language/en, 2018. Accessed: 2018-09-28.

[16] embien. Working with Automotive CAN Protocol. http://embien.com/blog/working-automotive-can-protocol/, 2018. Accessed: 2018-09-28.

[17] B. A. Forouzan. *Data Communication and Networking*, chapter Standard Ethernet (13.2), Cyclic Codes (10.3), pages 364–376, 264–277. McGraw-Hill Higher Education, fifth edition, feb 2006.

[18] F. S. Germany. *Formula Student Rules 2019*, 1.1 edition, oct 2019.

[19] R. B. GmbH. *CAN Specification Version 2.0*, 2.0 edition, sep 1991.

[20] M. Holdaway. *Designing antialias filters for ADCs*. Xignal Technologies AG, nov 2006.

[21] Infineon. *IFX1050GVIO - CAN transceiver*. Infineon Technologies AG, 1.0 edition, oct 2016.

[22] N. Instruments. Controller Area Network (CAN) Overview. http://www.ni.com/white-paper/2732/en/, 2014. Accessed: 2018-09-28.

[23] IXXAT. *CAN-to-USB compact*, 1.6 edition, sep 2011.

[24] kabeltec. *BUS-LD*.

[25] J. J. Li Tan. *Digital Signal Processing: Fundamentals and Application*, chapter Sampling of Continuous Signal (2.1), Realization of Digital Filters (6.6), pages 15–20, 192–199. Academic Press, second edition, 2013.

[26] NorComp. *CCA-000-MYYR234*, 1 edition, apr 2015.

[27] NorComp. *850-004-203RSSY*, 3 edition, mar 2016.

[28] NorComp. *852-003-113R00Y*, 4 edition, mar 2016.

[29] NorComp. *852-004-113R00Y*, 4 edition, mar 2016.

[30] M. I. Products. *MAX7400_MAX7404 - 8th-Order, Lowpass, Elliptic, Switched-Capacitor Filters*, 2 edition, jan 1999.

[31] M. I. Products. *MAX7401 - 8th-Order, Lowpass, Bessel, Switched-Capacitor Filters*, 1 edition, jun 1999.

[32] M. I. Products. *MAX7480 - 8th-Order, Lowpass, Butterworth, Switched-Capacitor Filters*, 0 edition, jan 1999.

[33] P. Richards. *A CAN Physical Layer Discussion*. Microchip, 2002.

[34] E. Tran. *Multi-Bit Error Vulnerabilities in the Controller Area Network Protocol*, may 1999.

[35] S. Vikings. *Suspension Report SDU Vikings*, 2018. Calculations of the suspension on the Viking X.

14 Overview of digital Appendix

1. **3D-model node**: A rendering of the PCB including layers, components and silkscreen.
2. **Electronics**: Schematics of electronics.
3. **Datasheets**: Datasheets for components.
4. **Articles**: Publications on CAN and similar.
5. **Tests**
   - Test journals
   - CAN logs and raw data
6. **Calculations**: Contains different examples of calculations. The folder contains the following:
   - CRC_calculator.m
   - FilterSpecifications.m
7. **Code**: libraries, drivers and source code developed.
8. **Summary of meetings**sf

15 Word list

- **ATmega32M1**: Atmel micro controller selected for use in this project.
- **SPI**: Serial Peripheral Interface.
- **ISP**: In-System Programming.
- **debugWIRE**: Single pin for asynchronous two-way communication with the micro controller which allows for debugging.
- **SCF**: Switched Capacitor Filter.
- **PCB**: Printed Circuit Board.
- **CAN logger**: Alternatively CAN sniffer. A device that can log and display messages sent on a CAN network. In this project IXXAT CAN-to-USB compact is used [23].
- **IDC**: Insulation-displacement contact.
- **SCS**: System Critical Signal.
- **CANL**: CAN low, this signal is driven towards ground when transmitting dominant bit.
- **CANH**: CAN high, this signal is driven towards supply voltage when transmitting dominant bit.
- **Integrator**: The person(s) on the SDU-Vikings team responsible for integrating the sensor network in the car.
- **High Voltage**: Defined in the Formula Student rule set as voltages above 60 V DC or 25 V AC. [18, p. 75]
- **SNR**: Signal to Noise Ratio.
- **Race conditions**: When a variable in a software program is at risk of being changed while in use.
- **CSMA**: Carrier Sense Multiple Access.
- **CAN controller**: A unit which processes and enables CAN communication.
- **CRC**: Cyclic Redundancy Check.
- **Shutdown circuit**: Required for shutting down the car. Reacts on SCS from the car