KTH Mechatronics Advanced Course

MF2063, HT 2018

FINAL REPORT

ESS-NW/ESS-CAR

Jonas Ekman
Yini Gao
Jacob Kimblad
Leon Fernandez
Fredrik Hyyrynen
Yifan Ruan

December 11, 2018

Abstract

Autonomous vehicles are extremely complex computing systems. A large amount of computing nodes are embedded in every vehicle in order to achieve the desired functionality and reliability. Due to the number of nodes reuquired, traditional on-board car communication schemes are no longer sufficient. Ethernet is emerging as an interesting alternative for use in communication between computers in autonomous vehicles due to its large bandwidth. Unfortunately, Ethernet is not optimal for use in time-critical systems, and in an autonomous car, this could be fatal. Software-defined networking seeks to solve the types of problems that emerges when using ethernet in a time-critial system. This report proposes a small scale prototype of an autonomous vehicle running a local ethernet network between four devices running embedded linux. The prototype incorporates five different services whose messages populate the network. These services has shown to pass 54 out of 84 tests that were aimed to fulfill the given requirements of the vehicle which included time-criticality aspects. The results shows a promising prototype but also that a lot more research needs to go into verification and validation of the time-criticality before such a system can be used in actual autonomous vehicles.

Contents

List of Figures

List of Tables

List of Acronyms and Abbreviations

- ESC : Electronic Speed Controller
- FIFO : First In First Out
- IR : Infra Red
- PCB : Printed Circuit Board
- PWM : Pulse-width modulation
- SDN : Software Defined Network
- SPI : Serial Peripheral Interface

1 Introduction

This report presents the process and results of two projects "Embedded Service for Self-adaptive Network" (ESS-NW) and "Embedded Service for Self-adaptive Car" (ESS-CAR). This chapter will start by describing the background of the two projects. The next thing to be described is formulation, goals and motivation of the two projects. Following this will be a short discussion on the delimitations for our team. The last part of this chapter will present an explicit report disposition which helps readers to get a sense of the overall report.

1.1 Background

Autonomous cars is a very popular field of study, within academia as well as the industry. In recent years, big companies such as Google, Amazon and Apple and Swedish companies such as Volvo and Scania have been investing heavily in this technology as it is expected to revolutionize the transport sector.

To ensure safe driving, a lot of sensor and actuator systems are embedded within the autonomous car. This allows it to make smart decisions and be adaptable to the dynamic circumstances it faces in the traffic. However, since basically every sensor or actuator is run by a computer it means that the complexity of the computer network aboard the car increases considerably. Traditional technologies for on-board car communication are no suited to handle this growing complexity.

Ethernet has been a staple technology and fundamental building block for the Internet for many years. Thanks to the large amount of time and research spent on Ethernet, it can provide data rates many times higher than that of traditional communication schemes for cars. Therefore, Ethernet is a promising candidate for providing the capacity needed for the autonomous car's network.

Ethernet is, unfortunately, not without flaws. It has been adapted for systems where capacity and scalability have been favoured before robustness and predictability. Therefore, it does not naturally live up to the demands of an autonomous car. By employing a Software-Defined Network (SDN), it might be possible to mitigate the shortcomings of Ethernet. A SDN employs a central controller that arbitrates all the communication in the car. The SDN controller can for example reroute packages and drop packages, all to ensure that high priority communication reaches its target at the right time.

Furthermore, Ethernet allows for sporadic connection and disconnection of nodes and thus provides no inherent way of detecting whether a node is missing due to, for instance, a faulty cable. Hence, the autonomous car needs to have some sort of way to perform a self-monitoring in order to ensure itself that the system status is acceptable for driving.

By implementing a prototype of a vehicle that uses a SDN controller and self-monitoring services to achieve a failsafe behaviour, this project aims to provide a proof-of-concept that ethernet can be used for communication in autonomous cars.

1.2 Project Description

The two projects "ESS-Car Embedded Services for Self-Adaptive Car" and "ESS-Network Embedded Services for Self-Adaptive Network" were provided by professor De-Jiu Chen at KTH Royal Institute of Technology. The idea of the two projects are from Viktor Karlquist's master thesis [1] where he presented a design and an implementation of an automotive experimental platform for ADAS. And the "ESS-Car" team from last year's MF2063 course has already implemented an autonomous vehicle prototype based on the thesis. Our projects are focused on the development and extension of the prototype.

1.2.1 Project Goals

The goals of the two projects could be divided into two parts, goals concerning the network implementation and goals concerning the remaining hardware and software system architecture. The overarching goal of the network implementation part was to configure the network using software defined network method which could help researchers explore what network implementations can be beneficial in this kind of autonomous vehicle infrastructure. The overarching goal of the rest part became to increase the robustness of the system and to implement additional vision services into the system.

1.2.2 Project Challenges

The main challenge we faced during the process of projects was to have too many nodes in the system which greatly increased the burden of debug and test. Another challenge was the data transfer between the local service and Vsome/Ip service in the Beaglebone. There was also a challenge coming from car assembly in which case every node shall be installed on the car and work smoothly.

1.2.3 Prototype formulation

Our final prototype fulfils most project requirements we proposed at the beginning stage while there are still some requirements missing. This section will only describe what we have implemented.

The prototype is able to measure current distance and speed data and detect defined flags. All nodes are mounted on the car and connected to the network. The network consists of SDN switches and controller is support network and the other network which is made up of Beaglebones is main network. Meanwhile, each of Beaglebones has its separate function, one for reading distance data and sending steering angle, one for reading speed data and sending target speed. These two Beaglebones act as SPI master in SPI communication who have two slaves each. The last Beaglebone acts as the system controller. Also a Raspberry is directly connected to the network providing object detection function. Four Arduinos are used to carry out specific functions, and communicate with corresponding Beaglebone through SPI. The prototype has two control modes, shell mode and automotive mode. In the shell mode, users could send commands from controller Beaglebone to other Beaglebones, then gather required information or change the status of the prototype. In the automotive mode, a state machine will run which try to maintain the speed of the prototype and avoid collision.

1.3 Delimitations

1.4 Report disposition

Chapter 1 describes the projects' background and gives an introduction to the two projects. Chapter 2 provides a short description of theories used in the projects. In chapter 3, The methods, tools and techniques used to solve the problems in the project are specified. The implementation of the autonomous vehicle prototype is presented in chapter 4. Chapter 5 describes how we verify and validate the prototype. The results of the two projects are shown in chapter 6, then chapter 7, has a discussion and a conclusion on the final results. The last chapter, chapter 8, lists some future work.

2 Literature Review and State of the Art

2.1 Network

2.2 Software defined network

Software-defined network (SDN) is a type of network where a controller decides how the traffic in the network should go. In a traditional network is the intelligence in the switches and they decided on what port the package should be sent out on. In SDN is a device called controller connected to all switches and monitors the traffic load on the links and find the most optimal path between node A to node B. The controller's task is to request information from the switches about what links are up or down and the traffic load with this information and decides how packages should be forwarded on the switch. Because the controller can monitor the topology of the network is it easy to scale the network with new nodes and switches[2].

2.3 Serial Peripheral Interface

Serial Peripheral Interface (SPI) is a synchronous serial interface specification for short distance communication. SPI supports full duplex mode using a master-slave architecture with one single master, and the SPI master originates the whole communication. The SPI bus has four logic signals: serial clock (SCK), master output slave input (MOSI), master input and slave output (MISO), slave select (SS). The detailed pin mapping of SPI master and slaves is shown in figure 1.

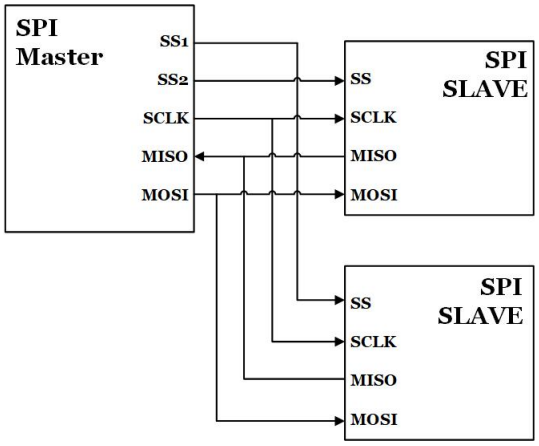

Figure 1: Connection of one SPI master and two SPI slaves

SPI has a higher throughput than those traditional transaction protocols, e.g. I^2C, UART, as it is not limited to any maximum clock speed enabling potentially high speed. And the hardware interface is pretty simple where slaves only use master's clock and are able to share the other three signals but the SS. Meanwhile, the software is not hard to implementation. For the opposite side, SPI has its limitations. There is no slave acknowledgement or hand-shaking mechanism in SPI, so the master could keep sending data to nowhere and not know it.

2.4 SOME/IP

SOME/IP is a client/server protocol originally developed by BMW. Its official website can be found at [3]. SOME/IP support two paradigms of communication, Request/Respond and Publish/Subscribe. The SOME/IP implementation used for the project, VSOMEIP, has a built-in service detection and multicasting mechanisms and allows the user to define the behaviour of the program based on triggers

such as the disconnection of a service, receiving a specific event message or seeing a service come online. VSOMEIP can be found and is documented at [4]

2.5 Assembly of the car

To make the car moving all the devices and components on the car has to placed on the car and because of there many devices on the car that communicates via SDN and has all of them be placed in an efficient way. To do that needs some kind of platform to be created to mount all the devices on and they are close to the other devices its depended on.

2.6 Power Supply

The car uses a 2 cell lithium battery what delivers a voltage of 7.2V. On the car is two different voltage level used, the first one is at 9V for a switch, and the second level is at 5V to power all the other components.

To generate a voltage at 9V from 5V or 7.2V has some kind of DC-DC converter be build to generate a higher voltage.

3 Methodology

3.1 Software defined network

The SDN switches used in this implementation is the Zodiac FX from Northbound Networks. They are switches made to support OpenFlow protocol and is designed to be used in SDN networks. On the switch is 4 ports there one port that has to be connected to the controller, all the other ports can be used to connect devices on. In the network is 4 devices connected and the switches has to be connected to each other has the network 3 switches.

Mininet is a Python-based application what an SDN network topology can be created and simulated to test that the controller works and analyse how the network will operate. The program also has the opportunity to connect the simulated topology to your physical SDN controller.

The car network topology was simulated in Mininet to analyse its behaviour, and to develop code for the controller. Under this test did we discover what that Mininet simulation had a much lower delay then the real implementation had, even then the extra settings on the links and nodes were added. This resulted that Mininet could not be used to analyse the best topology for the car. It was instead used to simulate the network to create a code for the Ryu controller. To analyse the Car network was two different Mininet scripts created, one was to use the physical raspberry pi controller and the second one was to use the internal version of Ryu in the Mininet VM. For the external simulation to work has the script be configured with the right IP-address to the raspberry pi.

3.2 Software Architecture

The driving system consists of several layers of software and communication. Figure 2 and 3 give a broad overview of how the software is structured. A custom driving software is running on the client BeagleBone. This can be either an interactive shell or a test suite of commands for testing and debugging basic functionality or a more intelligent autonomous driving software. The custom driving software controls the car through a library called ESSPrototype.

ESSPrototype is a C++ library that represent fundamental functionality for manoeuvring and monitoring the car. Some examples include controlling the steering, getting speedometer readings and checking if the distance sensors are online. Input and output to ESSPrototype is done via a UNIX shared memory and the IO packets can be seen in Figure 4. Table 1 contain brief descriptions of the packets.

At the other end of the UNIX shared memory that ESSPrototype uses, a client/server software runs and it is based on an application-layer protocol called SOME/IP [3]. The C++ implementation of the protocol used for this project was VSOMEIP [4]. The client BeagleBone uses SOME/IP to communicate over Ethernet with the two service BeagleBones who provide one actuator service and one sensor service each, namely distance sensing and steering, and motor control and speedometer. The Raspberry Pi that handles the camera also communicates with the Client BeagleBone in a very similar way using SOME/IP.

On the respective service BeagleBones, SOME/IP messages are transmitted and received. In fact, the message payload may be any packet in Figure 4 except the GO Status packet. The received messages are written to a UNIX shared memory and the payload for messages to be transmitted are read from a UNIX shared memory. On the other end of the shared memory, a task runs that handles the SPI communication with the Arduinos for the respective services, distance sensing and steering on one service BeagleBone and motor control and speedometer on the other.

Sensor Arduinos send measurement values to their respective master BeagleBone via SPI and actuator Arduinos read instructions from their respective master BegleBone over SPI as well. On each Arduino runs a piece of Arduino code that interfaces with the hardware using the Arduinos GPIO pins.

The camera communication works a bit differently. Firstly, it is a one-way communication since the camera is acts solely as a sensor. SOME/IP transmits a feature extraction that it reads from a standard UNIX pipe (—). At the other end of the pipe, a Python program running openCV feeds the extracted features into the pipe.

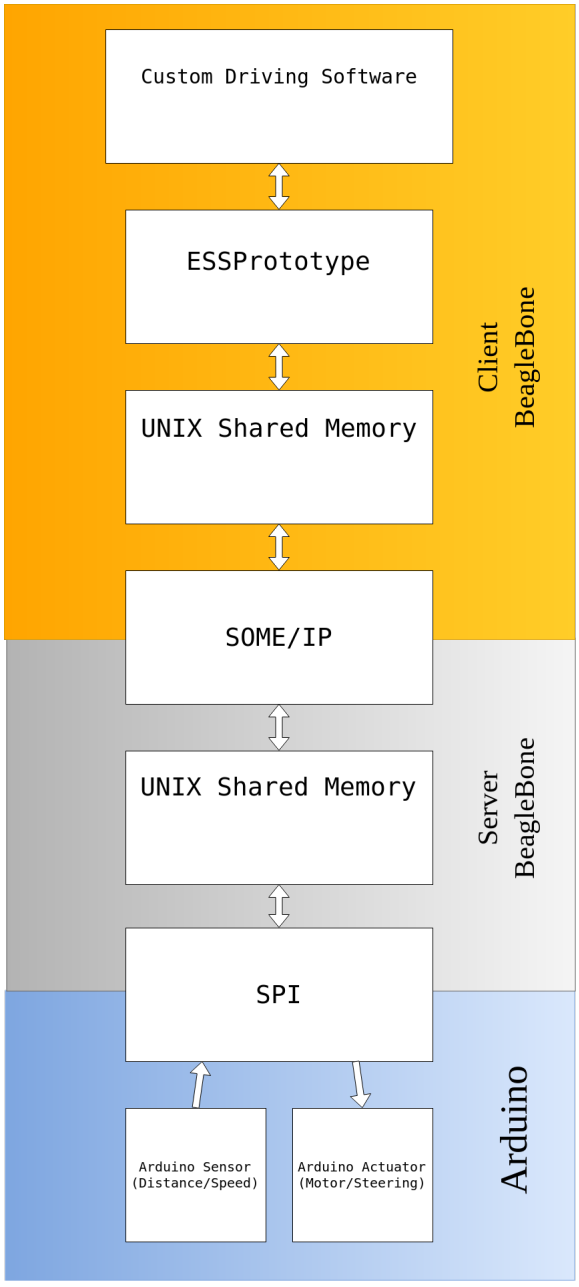

Figure 2: Overview of how the service Beaglebones are part of the software architecture.

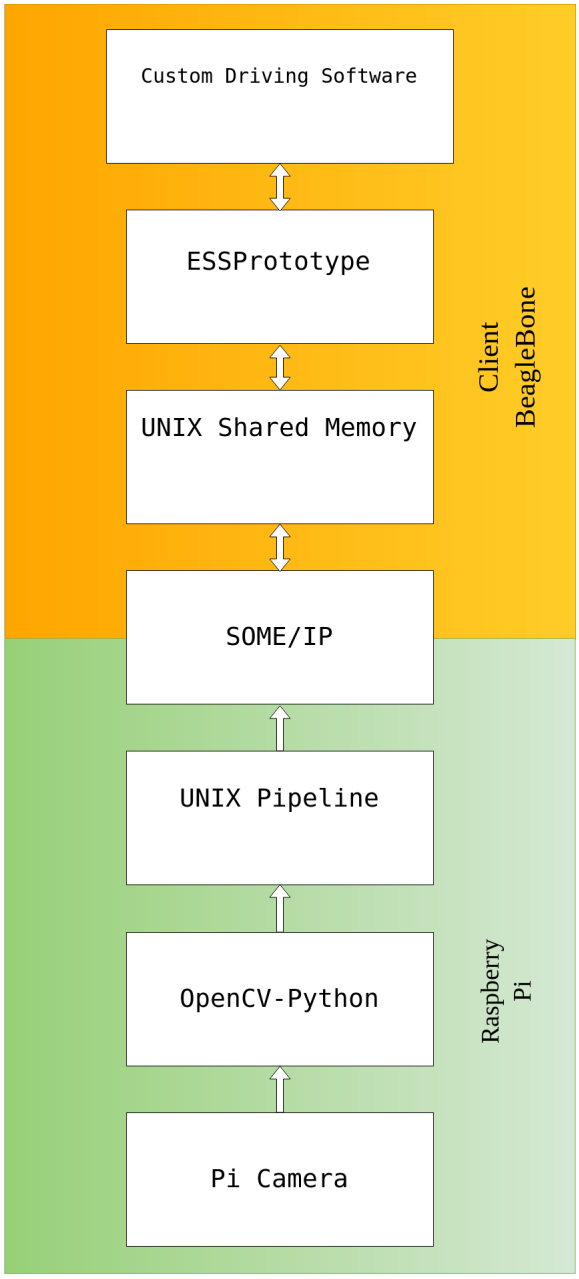

Figure 3: Overview of how the camera service Raspberry Pi is part of the software architecture.

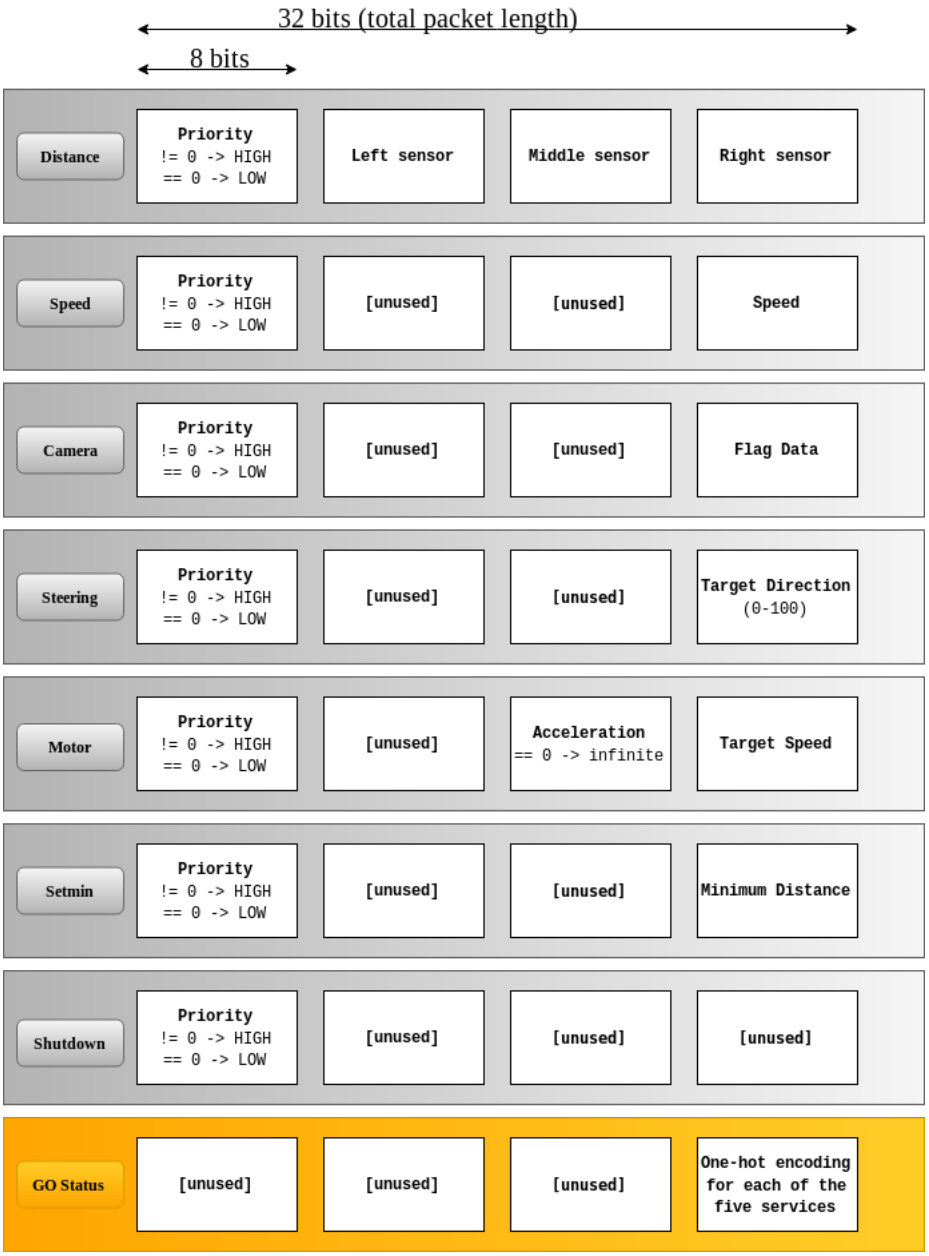

Figure 4: Input/Output packets handled by the ESSPrototype library.

3.3 Project management

Scrum project management is used during the process of our projects. The scrum framework has been adopted for use in this process because of it being agile as well as the fact that most group members had some type of previous experience or knowledge with it. Both software and physical tools were used for monitoring the projects progression. Excel was for example used to keep track of milestones, earned value and the resource plan and the online tool Trello was used alongside a physical whiteboard to keep track of scrum sprints and the project backlog as depicted in figure 6. A lot of design was also produced and kept on whiteboards as to promote discussion within the group and to keep everyone updated with updates or changes to design or models.

As with all projects where the team members have no previous experience in working with each other there will be a roll-seeking phase where the dynamic of the group is slowly figured out and

Table 1: Brief descriptions of the packets handled by the ESSPrototype library.

Type	**Description**
Distance	Measurements from the distance sensors (in cm)
Speed	Measurement from the speedometer (in cm/s)
Camera	Feature extraction from the camera containing colour and position information
Steering	Desired steering angle (0 means full left, 100 means full right)
Motor	Desired speed along with the acceleration for achieving that speed
Setmin	Set threshold value for when the motor should perform an emergency break
Shutdown	Packet that shuts down all services on a node, no data payload required
Go Status	Packet that shows what services are available (distance, speed, camera, motor, steering)

Risk nr.	Risk	Probability (P: 1-4)	Consequence (C: 1-4)	Risk value (P*C: 1-16)	Consequence Described	Proactive Measure	Proactive Milestone	Reactive Measure(s)	Responsible
1	Wipe working node	4	4	16	Correctly implemented nodes is mistakenly used for other purposes. Many nodes uses the same kind of hardware.	Put finished devices back in the box and mark box with sticker.	-	Fetch code from backup server and attempt to re-build the program on the node.	Everyone
2	Bugs	4	3	12	Some parts of the system behaves differently from the specifications. Other parts of the system that are relying on these faulty parts will fail or result in bad behaviour in some situations.	Always have one stable branch with tested and deployment-ready code. New features are merged into the master branch when finished and the master branch is merged into the stable branch when it has been tested for integration bugs.	Git repo has been started and rules established at the time of writing	Roll back the master branch to the latest stable version and step forward to investigate which commit was causing the issue then debug it.	Fredrik
3	Lack of knowledge	4	3	12	Lack of knowledge in some areas results in the inability to develop parts of the system correctly.	Sort tasks into independent fields (Electronics, Control etc.) so that no one member has to understand all the details of the system.	WBS has been performed at the time of writing	Responsible person reads up on the the missing knowledge. Other group members who get stalled move to other tasks in the meanwhile.	Fredrik
4	Unorganized code	4	3	12	Unreadable code or code that is hard to find makes debugging and validation hard to perform.	Set a couple of dedicated dates where the group sits down together and cleans and rebases the codebase.	<To be decided>	Hold an unplanned "emergency" session where the group cleans and rebases the codebase.	Leon
5	Unorganized documentation	4	3	12	Makes system validation and merging of system parts hard.	Use an automated documentation generator.	Doxygen has been chosen as a generator at the time of writing	Member who finds poorly documented code posts a note on the scrum board.	Leon

Figure 5: Extract of the risk analysis

which different positions the team members fit in. To make the group get into the process of working quickly the group decided to follow the results given by the belin test which was performed by all group members.

Setting up clear milestones for the project was very beneficial in management as it helped to divide the project into different phases in time with each one having clear expected outcomes. This helped the project stay on track with regards to time as it could easily be followed what needed to be done and when. As well as milestones the team also performed an in-depth risk analysis of the project and subject area. An extract of the risk analysis can be seen in figure 5.

To ensure smooth progress it was decided to have weekly meetings with the stakeholders were all group members could go through what they had done the previous week and receive feedback from the stakeholders. The feedback could be general tips and aid on the current assignment or that they wanted to steer a part of the project in a different direction or note that there might have been some part or parts that have been overlooked in the design or implementation. These meetings were also useful for raising issues that were encountered during the development of our project sponsors. The messaging app called Slack which is often used by development teams in the industry was also used by the project group including the sponsors. This allowed the group to bring up urgent issues with the project sponsors directly, without having to wait until the next weekly meeting and getting a quick response as soon as possible.

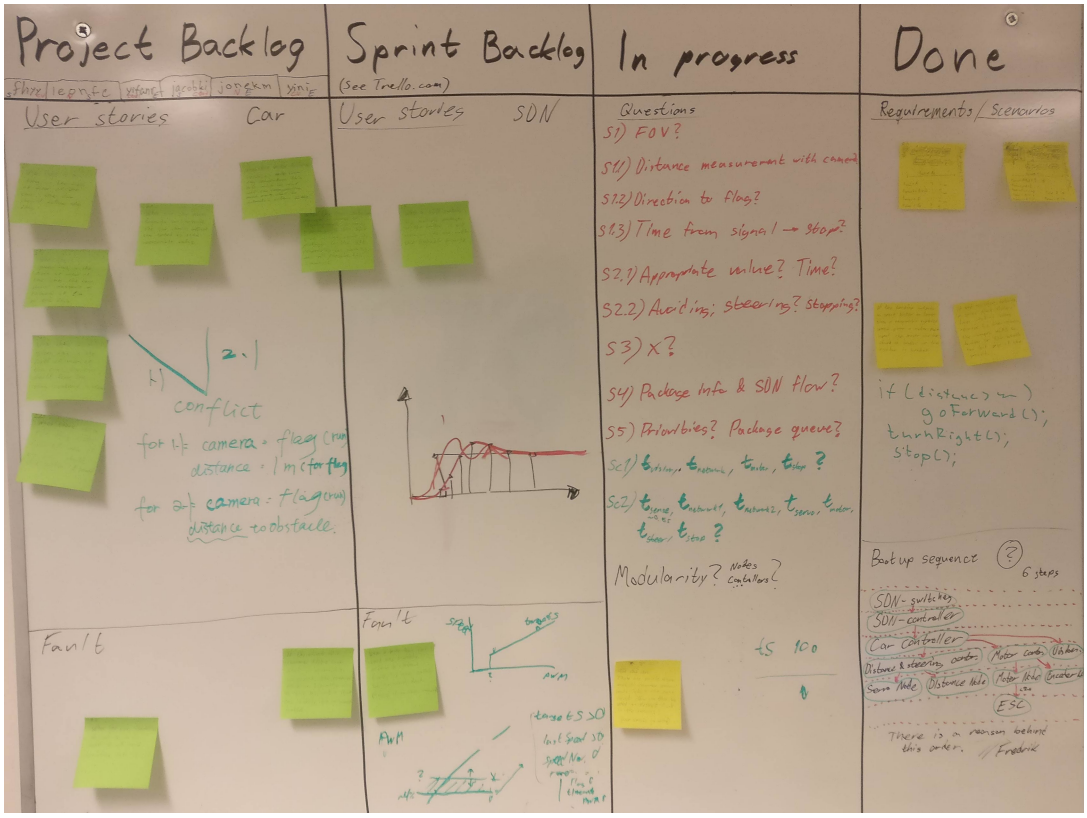

Figure 6: Scrum board continuously used throughout the project

3.4 Assembly of the car

The car platform used in this project is a car platform is Turnigy SCT 2WD 1/10 Brushless Short Course Truck (KIT) upgraded version and it was provided by the stakeholders at the start of the project.

To place all the required components on the car had some kind of platform be created to mount everything on. The requirements for the platform is if possible the car chassis should be able to fit on top of the car, all the device should be mounted on it and it should be easy to remove from the car.

To make this platform was it first draw in a Cad Fusion 360 to make a 3D model of the car. This model could then be used to add models of the devices used in the car to find the optimal place for them, so the are places in at an easy position to get access too and close to its devices its depended on. The result of the design is shown in figure 7.

3.5 Power Supply and PCB

The power supply for the devices on the car is done via two different levels one at 5V and the second one at 9V. This requires some converter to converts the 7.2V from the battery to 5V and 9v. The 5V DC-DC converter is a Turnigy USBEC-15A it as voltage regulator made especially for lithium batteries and has input voltage range between 6V-12.6V and it can deliver 5V volt as output. This converter was used because it was already provided in the material we received from the stockholders and because it indicates how much power there is left in the battery. The most important reason the converter module was used is because lithium batteries are sensitive can easily be damaged, so it was a smaller risk the battery would be damaged and affect the project development.

The second converter is to generate the 9V and this is done via a DC-DC converter called boost converter. The idea of the boost converter was it should be connected after the first converter and

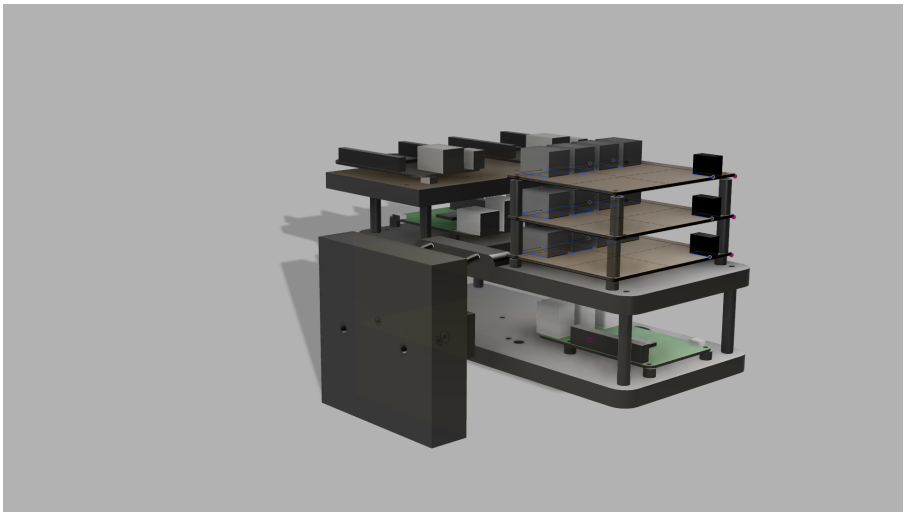

Figure 7: 3D model of the platform

would have an input voltage at 5V to generate a 9V output and a current of 0.6A. The reason it uses the 5V voltage level as input and not the 7.2V from the battery is to have all the powering done via the Turing converter and would not have something connected to the battery, that could damage it. The design for the boost converter was done in a program called MPLAB Mindi and is an analogue simulation program from Microchip, and it was used because the IC chip MIC2253 is from Microchip and it was provided as a PSpice simulation component so the circuit could be analysed. The MIC2253 is DC-DC boost converter wat has a switching frequency of 1MHz has an input voltage range between 2.5V-10V. The schematic for the boost converter is shown in figure 8 and designed form the recommendation provided in the datasheet for the MIC2253. The MIC2253 has the functionality of an Overvoltage Protection pin that is used to shit down the switch if the pin has a voltage higher then 5.6V, to solve this was to resistors R5 and R6 in parallel added to the OVP pin as shown in figure 8. The values of R5 and R6 is calculated from formula 1. To get the correct output voltage had R1 and R2 in figure 8 be changed to correct values. They are calculated from formula 2 and the V_{out} was set to 9V. This circuit was then simulated in MPLAB and it showed that the circuit should give an output voltage at 9V and a current peak of 0.9A, as shown in figure 9.

$$9V < 1.245 * \frac{67k\Omega * (R5 + R6)}{15k\Omega * R6} \tag{1}$$

$$V_{out} = V_{ref} * (\frac{R1}{R2} + 1) \tag{2}$$

All the powering of the car's devices should be handled by one PCB board, that contained USB ports and the boost converter to power the external switch. There the USB-A type ports should be used to power the Arduinos, Beaglebones, Raspberry Pi and Zodiac FX. For the powering of the external switch should a cable be connected from the PCB to the switch, it also to power outlets to power the PCB board for the Arduinos. This PCB board was designed in Eagle and the result is shown in figure 10.

For the Arduinos was two PCB boards made to easily connect the SPI connection between the Beaglebones and the Arduinos, and to connect external sensors. The first board was designed to easily connect the ultrasonic sensors and the steering servo with correct Arduino. The schematic and layout of this board was also made in Eagle and is shown in figure 11. The board is powered via a power via the DC power jack, to supply the Arduinos. On the top of the board is three connectors and they are for the Ultrasonic to be connected, and in the bottom is the connector for the steering servo. In

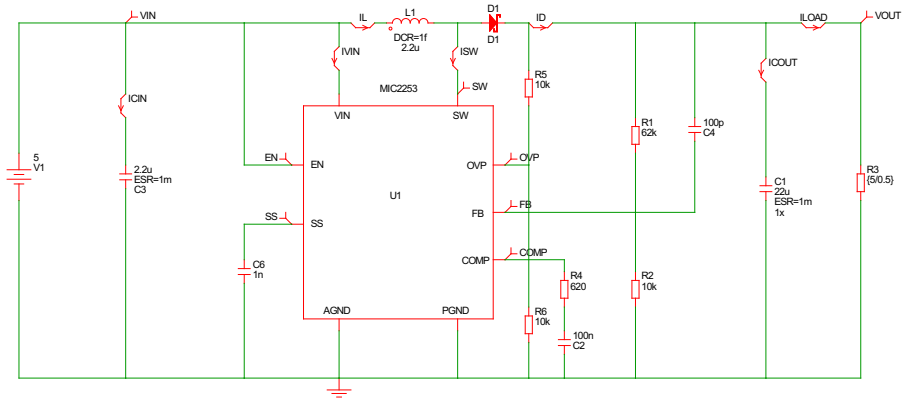

Figure 8: Circuit of the boost converter

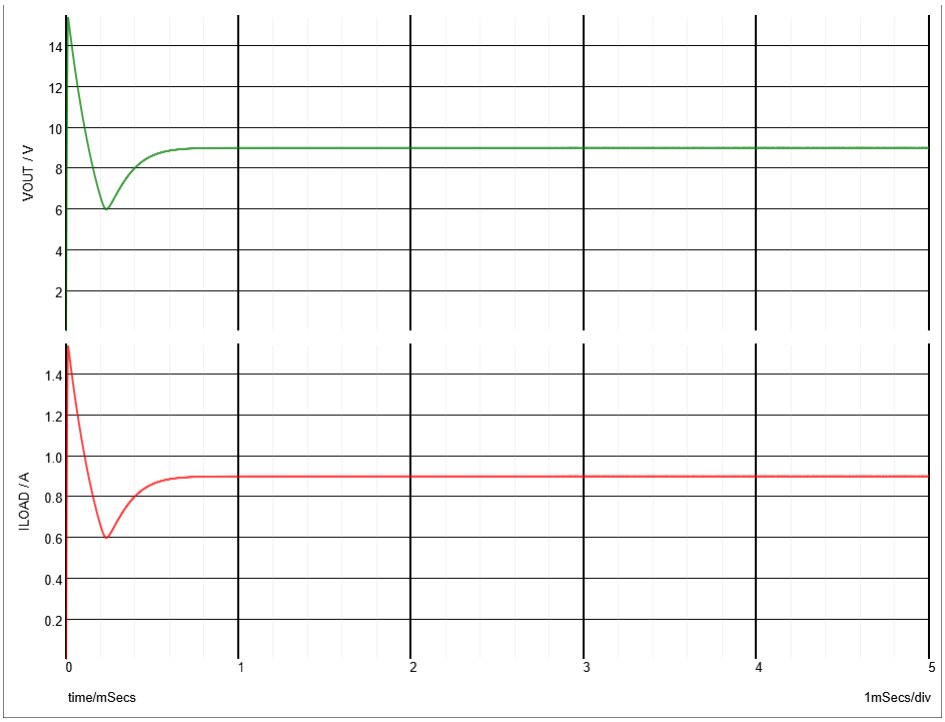

Figure 9: Output graph of the Boost converter

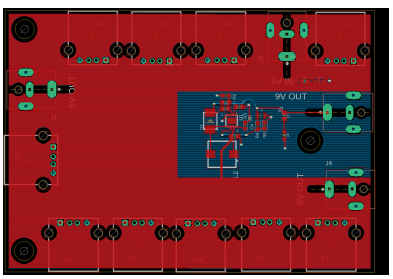

Figure 10: Layout of Power PCB board

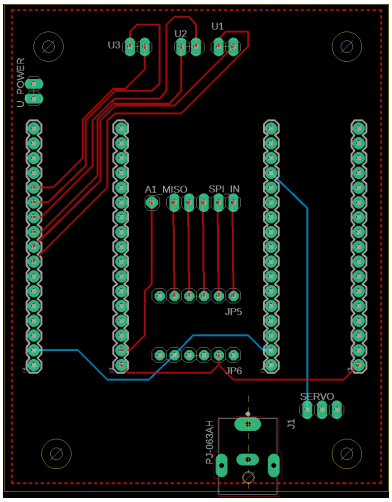

Figure 11: Layout of the Board for ultrasound and steering servo

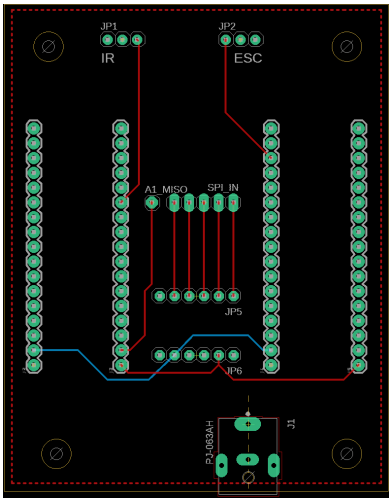

Figure 12: Layout of the Board for speed sensor and ESC

the middle of the board is the connection to add the Logic level converter and the pins to connect the connection from the Beaglebone.

The second Arduino board has the same functionality as the previous one. The difference is that this is designed to connect the speed sensor and the ESC to control the engine. The layout of this design is showed in figure 12.

4 Implementation

4.1 System overview

The overall architecture of our system is shown in figure 13. Basically, it presents how the hardware is assembled in our system.The cloud which consists of three SDN switches and Raspberry Pi is our SDN network, which also called support network in this report. The three SDN switches are all connected to each other, as well as to the SDN controller.

The System also have have three Beaglebone units and one Raspberry Pi. Two of the Beaglebone units are connected to two Arduino Micros each, so four Arduinos are assembled in the system. Theses Arduinos read sensor data or control the actuators and communicate with corresponding Beaglebone through SPI. Beaglebones connecting with Arduino receives the data from the Arduinos and posts it to the main network, Then the last Beaglebone, who acts as controller and subscribes the data, makes decisions based on current environment and then sends commands to other Beaglebones who will decode the command and transfer it to Arduinos connecting to the actuators. The last node Raspberry Pi with Pi camera is used to detect object in front of the prototype vehicle through which colour and position of the flag are able to be detected.

4.2 SDN network Implementation

At the beginning of the project was floodlight that SDN controller framework we decided to work with. Under the project way was a lot of problem coming up to integrate floodlight on raspberry pi due to some java packages used in Floodlight was not supported, and it had difficulties to communicate with the SDN switches. This resulted in that another SDN framework was selected, and it was a framework called Ryu. Ryu was selected because it was well documented with some example code to start with, and it was developed in python, so it should be no problem with integrating it on a raspberry pi.

The code for the Ryu controller is based on the example simple-monitor.py there the scripts request information from the switches about how many ports that are used and the amount received and send traffic. The modification don to it is to save the received information so it can be presented in on a webpage.

To monitor the traffic load on the network was a webpage made to present the user with data. The webpage is developt from the provided network webpage included in Ryu. On the webpage, as shown in figure 14 is a table what presented the data that has been transmitted and received on each port, and the number of packages has been dropped. The web pages are designed to update itself every 10s to be updated with the latest information. This has to do with the controller request new information from the switches every 10s. In the bottom of the webpage is a picture of how the network is configured.

4.3 Interprocess communication

To increase coherency and decrease coupling in the software architecture we chose to have several running processes on all of the Beaglebones. Each process is focused on doing a fewer amount of tasks which makes development easier. An example of this is the BeagleBone connected to the speedometer and the motor controller. It is running one process whose task is to talk to two Arduinos over SPI, where it sends information to the motor Arduino and gathers information from the speedometer Arduino. The other process on the BeagleBone is running the VsomeIP application. It is clear that these processes need to communicate with one another as the first process needs to access the ethernet.

The interprocess communication is implemented using shared memory. This is a POSIX concept which is implemented in Linux where more than one process can load in the same physical memory blocks into their own private virtual memory space. This is done using a few system functions and the operating system will make sure to take care of all the complex implementations.

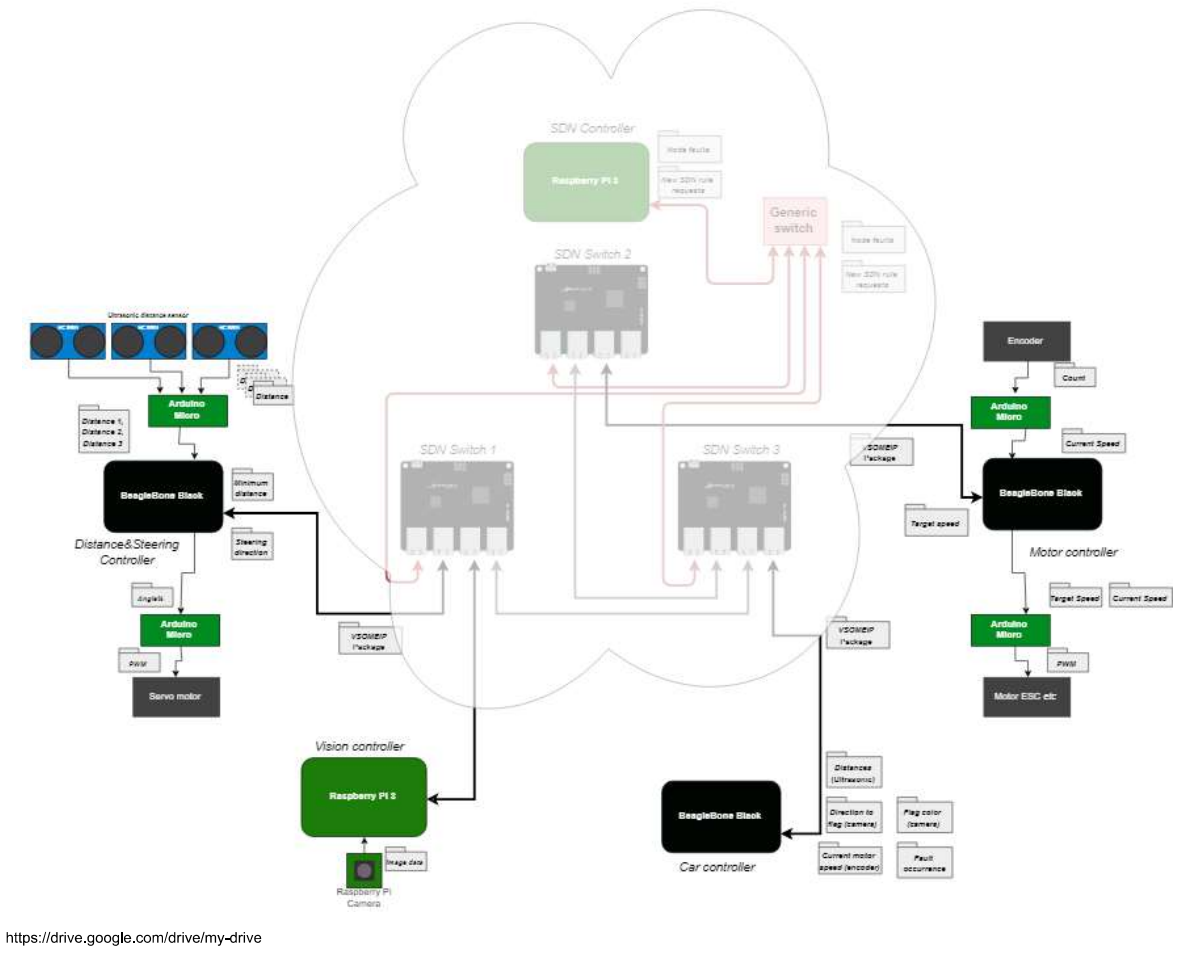

Figure 13: System overview

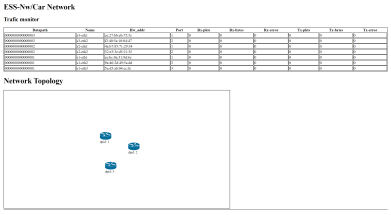

Figure 14: Webpage to monitor the SDN network

Using shared memory is not enough, the communication also needs to be defined such that all processes using the shared memory knows how it should be used. For the cars specific need a circular buffer is implemented on the shared memory. This is to have a type of flow control where the values in the buffer are always relatively fresh. If a FIFO buffer was used instead and it is not being read as often as it is written to the buffer might overflow, whereas now the oldest values in the buffer is just overwritten. When values are read from a FIFO the oldest value is always read first. There is no guarantee of how old this data is and might not be useful to read any longer. With a circular buffer it is possible to adjust the size such that the oldest value is always relatively fresh.

4.4 Communication between Beaglebone and Arduino

As the Beaglebone only supports to be the master in a SPI connection, so we have two Beaglebones with SPI configurations and each of them has two Arduino slaves. One Beaglebone has two SPI devices: SPIDEV0 and SPIDEV1, and the SPIDEV1 device is disabled initially but is used for HDMI interface.

In order to have one SPI device on Beaglebone connecting to multiple Arduino slaves, we use a third party C++ library called SimpleGPIO which enables us to use other general purpose pins as slave select of the SPI connection. Before we start communication with a specific Arduino slave, the corresponding slave select will be set to low, then we get access to that Arduino slave.

In the Arduino side, the communication request from the Beaglebone is treated as an interrupt. Then the Arduino writes/reads to/from the Serial Peripheral Data Register (SPDR).

4.5 Sensors

Three categories of sensors are implemented in the prototype vehicle to monitor its surrounding environments. Data from distance sensors and speed sensor will be sent to an Arduino initially, then sent to corresponding Beaglebone. Data from Pi Camera will be sent to the Raspberry Pi which is directly connected to the main network.

4.5.1 Ultrasonic sensor

The distance sensor used in the prototype vehicle is HC-SR04 which provides 2cm to 400cm non-contact distance measurement with accuracy to 3mm. Three HC-SR04 are mounted in our prototype in order to have a better scope of the distance measurement, as well as keep the prototype in a more safe way. To get data from HC-SR04, a short 10µs pulse should be supplied to the trig pin of ultrasonic sensor, then the sensor will send out an 8 cycle burst of ultrasound at 40 kHz and raise echo. The echo signal we get is a distance object which is pulse width and the range of the signal is in proportion.

An Arduino Micro handles both the generating the trig pulse and interpreting the echo signal. The Arduino will set the output pin to low, wait for 5ms, set the pin to high, wait for 10ms, set the pin back to low. This is the process of generating the trig pulse. After the Arduino sends out the trig pulse, it waits for 2 ms, then reads the value from the pin connected to sensor's echo pin. The last step is convert the received value to distance in unit of centimetre.

4.5.2 Reflective object sensor

A reflective object sensor consists of an IR-diode and an IR-receiver encapsulated in a housing and detects reflective material (such as white paper) placed perpendicular in front of its front. The sensor chosen was the OPB715Z from TT electronics and is characterised to detect white paper up to a distance of 12.7mm.

The reflective object sensor was used to implement the cars speedometer. It works as a type of RPM meter which measures the rotation of an object over a time period. It is then possible to extract

the speed of the rotation given the amount of rotation and the time period. To gather the amount of rotation the sensor is mounted on the axis of the wheel aimed towards the inside of the wheels rim. A number of strips of reflective aluminium tape have been mounted on the inside of the rim, with equal distance between them, to trigger reliable readings from the sensor. The physical setup is depicted in figure 15.

The sensor is connected to an Arduino using a Vcc, a ground and a data cable. The data cable continuously transmits digital information by either being logical zero equalling ground when no reflective object is detected and a logic one equalling Vcc (5V) when a reflective object is detected. It is therefore needed that the sensor is connected to an Arduino that processes this data further. The Arduino is triggered by an interrupt when the sensor detects a reflective strip attached to the wheel. The Arduino then fetches the amount of time since the last interrupt and calculates the current velocity.

One limitation with this approach is that the speedometer becomes less accurate the slower the car is going. An analysis of the case of standing still shows that a reflective strip inside the tire will never pass the sensor. This could be solved by allowing some amount of time to pass without detecting a strip before assuming that the actual speed is zero. It is however impossible beneath some speed to distinguish if the car is travelling very slowly or if it is standing still. To tackle this problem a total of ten strips were added to the inside of the rim such that strips will be detected more often thus increasing the accuracy of the speedometer at slow speeds. A new speed is also calculated each time a strip is detected which is something that has its trade-offs. It may result in more noise when travelling at higher speeds, but by instead waiting for more than one strip to be detected it becomes the type of running-average filter which helps to reduce noise that may be induced by an uneven spacing of the reflective strips.

Using the timing data and the physical properties of the wheel the velocity is calculated according to equation (3) where steps are the amount of reflective strips detected since the last calculation and the step distance is the physical distance travelled between two strips. The result can then also be manipulated to get the desired unit, which in this case was centimetres per second. The calculation is done on the Arduino before transmitted to a BeagleBone via an SPI interface. Since SPI transmits one byte of information at a time, and this specific implementation is limited to transfer no more, the maximum speed that can be measured is 255cm/s before the value wraps around.

$$velocity = \frac{(steps * stepdistance)}{time} \tag{3}$$

An idea for future implementation would be to dynamically adjust the amount of reflective strips that need to be detected before calculating the speed depending on the current speed to both increase accuracy when going slow and fast. Future implementations could also allow for more than just one byte of data to be transferred over the SPI, not limiting the max read speed without losing accuracy.

4.5.3 Pi Camera

In Pi's vision part, we tried to compare the performance of two different methods, which are object detection based on neural network and traditional computer vision technique using OpenCV.

To improve the accuracy of the deep learning part, we investigated if transfer learning techniques can be applied on the specific problem. On the other hand, to improve the real time performance, we think that model compression should be evaluated to fit the network into a smaller and faster net topology, without losing significant accuracy.

We have chosen several different pre-trained network to meet our demands, including shape-detection network and MobileNet. Shape detection network is a very simple and fast deep neural network which simply detect the shape of the object. But in the real test we found that this network is not robustness enough for our fast-moving scene since it frequently predicts wrong results.

Figure 15: Sensor mounted to the wheel axis on the left and the reflective strips mounted inside the wheel on the right.

Apart from using pretrained neural network, we have also train a Convolutional Neural Network to do the classification using online open source labelled dataset. The German Traffic Sign Detection Benchmark (GTSDB) is a labelled dataset for AI researchers who work on computer vision-based driver assistance. It contains 900 images, which divided in 600 images for training and 300 images for testing process. figure 16 and figure 17 are example images in dataset.

In the neural network that we trained by ourselves, we first down-sampling the images to 32*32 as input, then use 4 convolutional layers to extract the image features and use additional residual stack and dense layer to do the classification. The structure of neural network is shown in figure 18. The overall accuracy is good using these network because of the powerful prediction ability of DNN. However, the model is not that fast to deal with our car and it takes a lot of time for this network to process one video frame and produce output.

Then, we decided to use the technique of transfer learning to apply MobileNet on RPI, which is an efficient model for mobile and embedded vision applications. The speed remains slow although multiprocessing and hardware acceleration have been carried out to improve.

By using OpenCV's DNN module to apply a pretrained object detection network, we are able to pass input images through the deep neural network and get a output image with bounding box of specific object and label.

The distinct feature of MobileNet is that the core layer of it is depthwise separable filters. Depthwise separable convolution has a different structure from a standard convolution since the standard one put both filters and combines as inputs and get outputs in a single step, whereas depthwise separable one separate these two into two different layers, one of them as a filter and the other one for combining. This specific structure can largely reduce the computation cost and model size. The structure of these two convolution filters are demonstrated in figure 19 and figure 20.

The layer of standard convolution and of depthwise separable convolution are shown in figure 21 and figure 22. More details, figure 21 shows a standard convolution layer, following by batch normalization and ReLU. Batch normalization makes the learning of layers in the network more independent of each

Figure 16: Example image from dataset

Figure 17: Example image from dataset

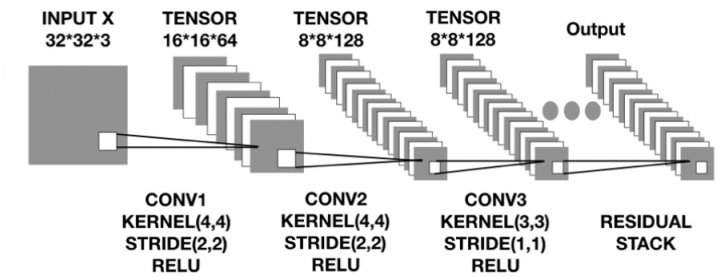

Figure 18: Structure of network

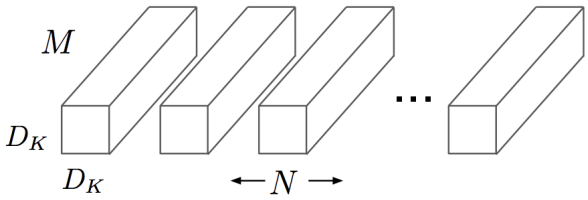

Figure 19: Standard convolution

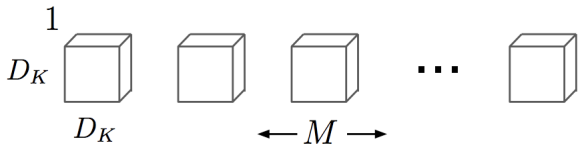

Figure 20: Depthwise separable convolution

other. It is a technique to address the problem that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. The technique consists of adding an operation in the model just before the activation function of each layer. Then add a ReLU activation function, which is rectified linear unit to define the positive part of its argument and negative part of 0. Figure 23 shows the plot of ReLU.

Figure22 describes depthwise separable convolution layers with 3 x 3 depthwise convolution and 1 × 1 pointwise convolution following by batch normalization and ReLU activation function after each of the convolutional layer.

Additionally, in order to optimize the Raspberry efficiently and use sufficiently limited resource and memory on it when running neural network, we did some optimizations. We firstly applied hardware optimization which is to install optimized OpenCV compile. ARM NEON, as an optimization architecture extension for ARM processors is installed. It is used for faster video stream processing, image analysis and speech recognition that is exactly what we are look for in our application. This architecture can execute multiple processing in the pipeline by a single instruction. Besides, VFPV3 as a floating point optimization is also used. After the first stage improvement, we get a 30 percent speed increase since we have make full use of the 4× ARM Cortex-A53, 1.2GHz processor on Raspberry Pi, now the network is able to classify an object in 1.5 seconds.

When it comes to second stage of optimization, multiprocessing is used to increase the speed of processing video stream. The I/O tasks, dislike CPU bound operations, always take lots of time and delay the process. So moving the reading of frames to a separate thread from processing the frame can obtain higher speed. Otherwise, every time I/O port access Pi camera, the main thread of script is blocked until the frame is captured and return to script. Multiprocessing can decrease the influence

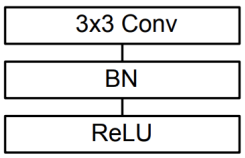

Figure 21: Standard convolution layer

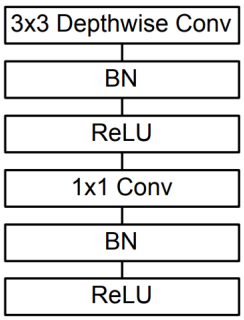

Figure 22: Depthwise Separable convolution layers

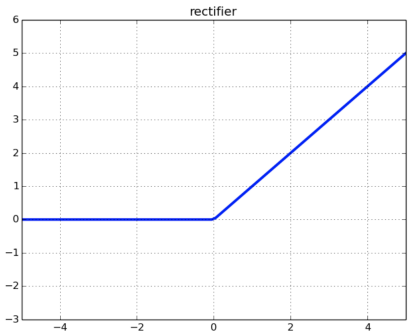

Figure 23: Plot of ReLU activation function

of I/O on CPU heavy application like video stream processing, especially in our real-time case. Now we can obtain a classification result in 1 second.

Moreover, there is a tradeoff between accuracy and output speed. In our application, we set a threshold for the output, in more details, only when the confidence score of detection result is high enough (above the specified threshold), the result can be output as a signal to steer the car. Otherwise, it will grab another frame and do the object detection for the other iteration. As a result, if the threshold confidence score of output increase, the output speed will decrease.

After several tests, we found that it is not feasible to use DNN on the RPI because of the predicting speed, RPI has really limited computational capability and has no GPU that can be used to do the DNN-accelerate. After the mid-term, we decided to try another strategy which using the traditional computer vision technique offered by OpenCV.

Then we try to process video stream by using OpenCV. OpenCV has offered many traditional and start-of-art computer vision techniques which can be used . We tried several algorithms for the detection in our specific scenario, including SIFT/SURF key point detection, mog2 background subtraction, hough transform, and colour detection using HSV space. Among these methods, we eventually chose colour detection as our camera detection algorithm.

The colour detection algorithm can be separated to different phases. First, we need to do the image pre-processing: we down sampling the images using different scaling factors since we want both processing speed and accuracy optimized. Then we transfer the RGB colour space to the HSV colour space to make the colour easy to be identified. We define the upper and lower limits for pixel values to classify three colours. Then specify which pixels fall into specified upper and lower range by masking. Finally, we do the detection by comparing the HSV value of our processing frame and the pre-defined HSV colour space value and output both the colour and position information. The speed of colour detection is fast and also accurate enough for real-time application.

To enable our model communicate with the VsomeIP perfectly and precisely, we tried several methods to connect Python program and C++ program, including using Google protocol buffers and RAM pipe transfer. We have to ensure that both of programs can communicate simultaneously without block.

Apart from outputting the detection results and communicate with the SDN network, we have also tried to send the video stream through WIFI or Bluetooth to Android application so that we can know what the camera sees through smartphone. Although the real-time video stream can be sent to a web browser, the sending script cannot run when the detection script is running since they will block each other.

4.6 Controlling actuators

4.6.1 Steering servo

A servo motor was connected on the front axis of the car to enable steering. The servo is connected to an Arduino using a Vcc, a ground and a data cable. To control the servo motor, a pulse of varying length should be applied on the data cable. A Arduino library for servo controllers was used as it has definitions for these pulse lengths and was compatible with the servo motor used in the project.

One issue with the servo motor mount was that a direction change required the servo motor to turn a bit before it had any effect on the steering axis. This issue was fixed by changing the software implementation such that a direction change added or removed from the output pulse length.

4.6.2 Motor ESC

The motor mounted on the car is a *TrackStar 17.5T "Outlaw" Sensored Brushless Motor V2* connected to a *HobbyKing® ™ X-Car 45A Brushless Car ESC*. The ESC is connected to an Arduino using the RX signal and ground cable only.

There are a few steps needed to setup the ESC before it can be controlled by the Arduino. For the set up in this project, follow these steps:

- Connect the RX signal and ground pin to the Arduino
- Load an Arduino program that allows you to manually send PWM signals on the signal pin. The program in mind uses the Arduino servo library function *writeMicroseconds* and the PWM signal values will be defined according to the input values of that library function.
- Start the Arduino program but keep the ESC off
- Send a PWM signal of 2000
- Hold down "set" button on the ESC switch and switch it on while still holding down the button
- Wait for a solid orange light
- Release the "set" button on the switch and wait for a solid red light
- Send a PWM signal of 700
- Wait for a solid orange light
- Send a PWM signal of 1000
- Wait for a beep and for the lights to turn off
- Switch off the ESC

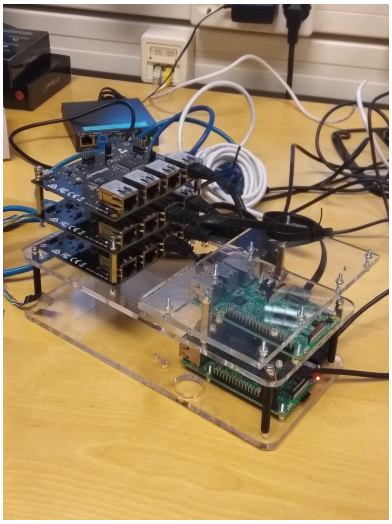

Figure 24: Platform

- The set up is complete. When switching on the properly set up ESC, make sure that there are no PWM signals higher than 1000 sent on the data pin.

The ESC has a safety measure for when the motor PWM is set to anything higher than *neutral* (a pulse width of around 1000 microseconds), and the motor is too weak to carry the load and thus will not spin. In this case, the ESC will not listen for any other PWM signal other than *neutral*. Once a *neutral* PWM signal is sent for a couple of milliseconds, the ESC will start handling higher PWM signals again. The Arduino program running the motor has means of dealing with this case.

The Arduino program reads data for target speed and current speed through an SPI interface. This interface sends one byte at a time, where the most significant bit defines if the data is for target speed or current speed. This data structure puts a limit on the values as there are only 7 bits left for data. The data is defined as 2 centimetre per second to increase resolution; sending a 10 would mean a speed of 20 centimetres per second. This allows the controller to read and maintain speeds of at most 255 centimetres per second with the only limit being the frequency of the speed sensor.

The Arduino program has two possible implementation of speed keeping. To switch between these, one has to change a line in the source code. One implementation is a direct translation from target speed to motor PWM. If the PWM requirements for reaching a speed would change, this translation would not longer be valid. Driving on a flat surface would still be fine. The other implementation uses a PID controller which reads the current speed from the SPI. As the closed loop with data through SPI over two nodes, the data will be delayed by a couple of milliseconds, but tuning the PID parameters can result in a good PID controller.

A future implementation could focus on lowering the delay over SPI and tune the PID parameters properly. Sending multiple bytes over SPI at once would also remove the need of scaling the data before sending, and increase the resolution by at least one bit.

4.7 Assembly of the car

The platform pieces are achyle plastic and were cut via a laser cutter, the schematic for the pieces was exported from the 3D model to Illustrator to be sent to the laser cutter. The result of it is shown in figure 7. The platform was mounted via board spacer of different sizes to mount the levels together. All the devices on were mounted via screws and nuts to keep them in place.

The design of the platform was done to use PCB board and not prototyping boards, this made the assembly of the car not to be as the idea was. This is because it was difficult to make the prototyping

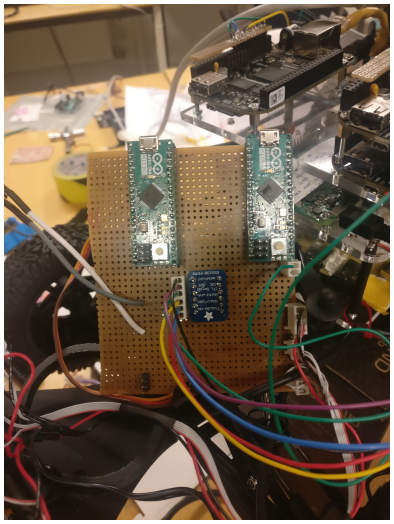

Figure 25: Arduino board for the steering and Ultrasound

board small and compact as you can do with PCB. Two of the boards the power board and one of the Arduino boards were able to place in the targeted position but could not be mounted to the platform with screws but with cable ties. The other board had to be placed in front of the car due to it could not be fit at its target place because the power board and the Arduino board was too thick of the wiring on the boards.

4.8 Power Supply and PCB

The idea of was to order the PCBs from a company, but due to course regulation ware we not allowed to order PCBs from China, and companies in Europe were too expensive to order from. This resulted in what we could not implement the boost converter we hade design due to the milling machine in both prototype centre and in Mentorspace was broken and they could not mill the layout for the boost converter. Because the milling machines were broken could the PCB for the Arduinos also not be made. The only way we could continue with was to make the design on a prototyping board.

The implementation on the prototype board was built from the same concept the PCB should have had, with USB-A ports cables to power the Arduino boards and external switch. The board connects via a cable from the battery to a fuse, this fuse is implemented to protect the system if a short circuit happens. To solve the boost converter circuit did we buy a boost converter module MT3608 to receive a 9V. This module is not connected to the 5V level but instead to the to the 7.2V from the battery, to receive a current higher than 0.6A. To prevent previous years failure then they used the module, did we add the same solution to add an led with a resistor in series and $6.8\mu F$ in parallel to the output pins on the boost converter.

The Arduino boards hade also be made at prototyping board. They received the same functionality as the PCB version should have, but it takes larger space due to the cable wiring. This lead to one of the board could not be placed at its position on the platform.

The result of the prototyping is shown in figure 25, 26 and 27. All of them has the functionality has the same PCBs should have had.

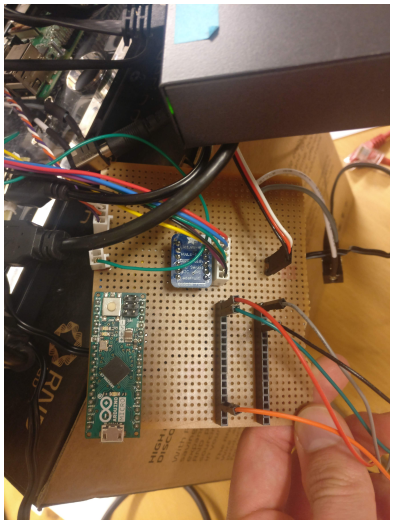

Figure 26: Arduino board for the engine and speed sensor

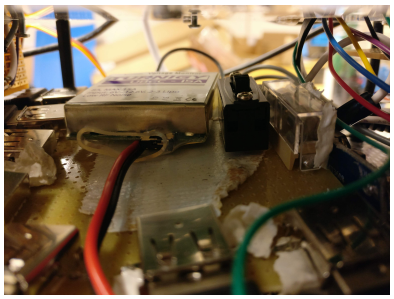

Figure 27: Power board

5 Verification and Validation

5.1 State machine

The state machine was modelled and verified using the tool NuSMV which is a symbolic model checker that is used for formal verification of finite state systems. Since NuSMV allows for the models to be checked using linear temporal logic the state machine could be checked for node coverage and edge coverage. NuSMV also allows for exhaustive analysis meaning that each set of combinations of the state variables needs to have a defined behaviour in each state of the model.

5.2 Requirements and functionality

The requirements and some of the system functionalities has been tested by following a set of test protocols and comparing the results with the expected outcomes (see Appendix A and B). The protocols covers the communication between sensors and Arduinos, Arduinos and actuators, the SPI communication between Arduinos and BeagleBones, and the VSOME/IP packages to and from BeagleBones. The requirements tested was the flag following system, the collision avoidance system, and some of the fault states of nodes.

Each node and controller were developed and tested continuously to ensure that it sends and receives data properly. There were no unit tests implemented due to timing constraints and the small scale of each individual component. The integration was the hardest part, and therefore the set of testing protocols found in the appendices A and B was created, to ensure that the functionality could be verified.

6 Results

The results of the project can be summarized by what requirements that were fulfilled. As the testing protocols in Appendix B are directly derived from the functional requirements it gives a good indication of the results of the project. The first table depicts the requirement that the car should be able to follow a flag. This requirement was not finished as the tests didnt pass due to the state machine and vision not being integrated properly.

Another requirement is collision avoidance. This requirement was met partially as the car was able to stop when detecting an object of a distance less than one meter from the front of the car. Because of the state machine and vision not being implemented properly the car was unable to detect objects on the sides and turn in the other direction.

The car is supposed to be failsafe, which is another requirement whose test protocols are depicted in Table 11. The test cases shows that the car is able to detect faulty states such as disconnected ethernet cables. It is not however able to discover disconnected SPI cables as this is shown to not have been implemented. The test cases shows that the car is reacting slightly slower than 2 seconds for the ethernet cables however, but commented that the test were run using debug compilation which produces a lot of prints thus worsening the response time.

It was a requirement to have complete working implementations of all the seperate services/nodes on the car such as steering, distance sensoring, speedometer, motor controller and computer vision/vision controller. The test protocols for these requirements are found in Appendix A, tables 2-8. As can be seen the steering node passed all the tests and was thus sufficiently implemented. The distance sensor also passed all of its functional tests with some notes that the received data is a bit noise and could use some calibration. The speedometer node passed all of its tests without any further comments and was thus implemented correctly. The motor node also passed all of its tests with notes on four out of eight tests that its parameters need tuning.

There were also requirements to implement nodes running embedded linux on BeagleBones to act as an interface between the Arduinos running actuators and sensors to the ethernet network. These had some requirements that the messages should be forwaded correctly both over SPI and over the ethernet network. The BeagleBone interfacing the distance and steering nodes passed all of its test as seen in Table 6 and was thus implemented correctly. The BeableBone interfacing the motor and speed nodes also passed all of its tests as seen in Table 7 and was thus also implemented correctly.

7 Discussion and Conclusion

7.1 Assembly and PCB

Due course regulation was we not allowed to order PCBs from China and to order it from a European company was too expensive. The idea for the project was to design our own PCBs because the project should combine the knowledge we hade learned during our masters and some of the students in the project is taking the electronics track. But due to the course regulation were we not allowed to order from companies that did not accept invoice as a payment, it resulted that we could not order from the cheap PCB manufactures in China. The group tried to find a European company but they were too expensive. This lead to a redesign of the powering and we decided to use the milling machines but they there also broken so the only solution was to use prototype boards. There is kind of strange that there is no machines or ways to order PCB because the electronic track is to about design the electronics and there is no way for us to be able to satisfy the course goals.

Because we hade to use prototyping boards and the platform was designed to have PCB mounted to it and not prototyping boards. This led to the assembly of the car was not as optimal as it could have been and the placement of come components and boards had to be placed at a different location. If the car would have been able to move around all it would have been affected of this, due the cable wiring was not optimal and the reliability of connections and wound have affected the car's performance.

7.2 Software Architecture

The main backbone of the software architecture seems to run well. It is well commented, structured and documented and should provide a basis for anyone who wishes to extend upon the project. Thanks to VSOMEIP, some basic system monitoring can be performed and corresponding failsafe measures can be taken. The software driving the car has access to see if any of the service BeagleBones or the camera Raspberry Pi gets disconnected and can thereafter react accordingly. And should the BeagleBone that runs the motor service somehow get isolated, it notices this and stops the car.

There still remains quite a lot of work to be done, however. Due to time constraints, very little effort was spent on fine-tuning the periods of the different tasks in the system. The responsiveness of the system is heavily dependant on the periods of the tasks and thus also the performance.

7.3 Test suite

The most important features of the system were tested with the protocol in appendix A and B. Most tests went okay with room for improvement while some shows poor performance. The test cases that failed were additional features that would be nice to have in a fully running system, but due to time constraints, these had to be put on a lower priority. The passing test cases all show that the requirements on the system were achieved.

8 Future Work

1. The Pi camera can also output the distance, so that it can work as an alternative of the ultrasonic sensor. Since we are attempting to build a fault tolerant and self-adaptive system, we need the car be able to get the required data even when some sensors break down. Additionally, we can also take the distance into account when steering the car. For example, only when the distance from the car to the signal up to a specific value, the car start to follow the signal. Otherwise, it receive the signal but not carry out steering.
2. Improve the communication between the colour detection python code and VsomeIP, which causes some delay in our current version application. One possible solution is convert the python code into C++, so that the communication protocol will be simple and feasible.
3. Although we have already tried to send the camera video stream to Android phone through WIFI so that we can directly see what the car is facing during running. We failed since the scripts of sending stream and colour detection always block each other, it would be feasible to copy the video stream captured from Pi camera and one of them used as colour detection, the other one can be sent to phone.
4. A more advanced monitor of the controller and lets the user to modified the network from it.

References

[1] V. Karlquist, "Design and implementation of an automotive experimental platform for adas," KTH, Skolan för informations- och kommunikationsteknik (ICT), 2017.

[2] K. Benzekki, A. El Fergougui, and A. Elbelrhiti Elalaoui, "Software-defined networking (sdn): a survey," *Security and Communication Networks*, vol. 9, no. 18, pp. 5803–5833. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sec.1737

[3] "Scalable service-oriented middleware over ip, official webpage," https://web.archive.org/web/20180805015646/http://some-ip.com/, accessed: 2018-08-05.

[4] "VSOMEIP, Official github repo," https://github.com/GENIVI/vsomeip, Version: 2.10.21.

A Functionality testing protocols

Table 2: Steering node

Action	Expected outcome	Results (2018-12-11)
Connect and start affected nodes properly	—	—
Send correctly formatted data via SPI interface to node	Data transmitted (Can be verified by adding logging and reading logs via the serial port of the steering node)	Ok
Send angle value 0	Front wheels turns 20 degrees to the left	Ok
Send angle value 50	Front wheels centres	Ok. Off by a small angle to the right
Send angle value 100	Front wheels turns 20 degrees to the right	Ok
Send angle value 50	Front wheels centres	Ok. Off by a small angle to the left
Send angle value 100	Front wheels turns 20 degrees to the right	Ok
Send angle value 0	Front wheels turns 20 degrees to the left	Ok
Send angle value 50	Front wheels centres	Ok. Off by a small angle to the right

Table 3: Distance node

Action	Expected outcome	Results (2018-12-11)
Connect and start affected nodes properly	—	—
Read correctly formatted data via SPI interface from node	Data received (Can be verified by adding logging and reading logs on the receiver node)	Ok
Hold a solid object 1 decimetre in front of the left distance sensor, but not the centre or right one	The node sends distance values 10, X, Y where X and Y are any distances sensed by the middle or right sensor. means within an error margin of 2 cm	Ok
Hold a solid object 1 decimetre in front of the right distance sensor, but not the centre or right one	The node sends distance values X, Y, 10 where X and Y are any distances sensed by the left or middle sensor. means within an error margin of 2 cm	Ok. Received data is a bit noisy
Hold a solid object 1 meter in front of the car	The node sends distance values 100, 100, 100. means within an error margin of 20 cm	Ok. Needs calibration, but sensors agree on a distance
Keep the area from 0 meter to 1 meter in front of the car clear	The node sends distances >100, >100, >100	Ok

Table 4: Speed node

Action	Expected outcome	Results (2018-12-11)
Connect and start affected nodes properly	—	—
Read correctly formatted data via SPI interface from node	Data received (Can be verified by adding logging and reading logs on the receiver node)	Ok
Keep the wheel still	The node continuously sends a speed value of 0	Ok
Start turning the wheel at some approximate rate	The node sends speed value of X cm/s where X is the distance in cm travelled per second if the wheel would turn at the current rate and be placed on the ground	Ok
Start turning the wheel at some other approximate rate	The node sends speed value of X cm/s where X is the distance in cm travelled per second if the wheel would turn at the current rate and be placed on the ground	Ok
Keep the wheel still	The node starts to continuously send a speed value of 0 within one second	Ok

Table 5: Motor node

Action	Expected outcome	Results (2018-12-11)
Connect and start affected nodes properly	—	—
Send correctly formatted data via SPI interface to node	Data transmitted (Can be verified by adding logging and reading logs via the serial port of the motor node)	Ok
Send target speed value 10 with the current speed value 0	The motor starts running and runs faster and faster with time, but stops before it reaches less than half of its maximum power	Ok
Stop, connect to a fully functioning speed sensing node and add speed data to the data sent via SPI, and start the motor node	The speed sensor starts logging values in cm/s	Ok
Send a target speed value of 10 with the measured speed values and continuously send measured speed values and the target speed values as soon as they are updated by the speed sensor node	The motor starts running and modulates itself until the speed of 10 cm/s is read by the speed sensor and maintains this speed after at most 1 second of modulation	Ok. Parameters needs tuning
Send a target speed value of 0 with the measured speed values and continuously send measured speed values as before	The motor slows down to a halt in less than a second	Ok. A bit longer than a second
Send a target speed value of 120 cm/s with the measured speed values as before	The motor starts running and modulates itself until the speed of 120 cm/s is read by the speed sensor and maintains this speed after at most 1 second of modulation	Ok. Parameters needs tuning
Send a target speed value of 20 with the measured speed values as before	The motor slows down and modulates itself until the speed of 20 cm/s is read by the speed sensor and maintains this speed after at most 1 second of modulation	Ok. Parameters needs tuning
Send a target speed value of 0 with the measured speed values and continuously send measured speed values as before	and continuously send measured speed values as before The motor slows down to a halt in less than a second	Ok.

Table 6: Distance-Steering controller

Action	Expected outcome	Results (2018-12-11)
Connect and start affected nodes and VSOME/IP clients properly (assume functioning distance node)	—	—
Read SPI messages and messages on the VSOME/IP protocol for distance data, sent by the node	A stream of distances correlating to the closets obstacle in front of the car will be read	Ok
Put something in front of the car at some approximate distance	A stream of distances correlating to the closets obstacle in front of the car will be read on the VSOME/IP channel for distance data	Ok
Repeat last step to verify it works for a set of approximate distances	—	—
Send a value of 0 over the VSOME/IP protocol for steering	The SPI channel for steering will read 0x00	Ok
Send a value of 100 over the VSOME/IP protocol for steering	The SPI channel for steering will read 0x64 (100 in decimal)	Ok
Send a value of 50 over the VSOME/IP protocol for steering	The SPI channel for steering will read 0x32 (50 in decimal)	Ok

Table 7: Motor-Speed controller

Action	Expected outcome	Results (2018-12-11)
Connect and start affected nodes and VSOME/IP clients properly	—	—
Read SPI messages and messages on the VSOME/IP protocol for speed data, sent by the node	A continuous stream of 0x80 will be read (due to the first bit acting as a data type id)	Ok
Prepare a byte 0x10 to send over the other SPI channel to the node	The first SPI channel will read a continuous stream of 0x90 (due to the first bit acting as a data type id). The VSOME/IP message for speed data will read the value 16	Ok
Send a value of 0x20 over the VSOME/IP protocol for target speed	The first SPI channel will read 0x20 followed by a continuous stream of 0x90 (as before)	Ok
Prepare a byte 0x00 to send over the other SPI channel to the node	The first SPI channel will read a continuous stream of 0x80 (due to the first bit acting as a data type id). The VSOME/IP message for speed data will read the value 0	Ok
Send a value of less than 0x16 (30 in decimal) on the VSOME/IP protocol for distance	The first SPI channel will read a 0x00 within less than a second	Ok
Send a value of more than 0x64 (100 in decimal) on the VSOME/IP protocol for distance	No change	Ok
Send a value of 0x20 over the VSOME/IP protocol for target speed	The first SPI channel will read 0x20 followed by a continuous stream of 0x90 (as before)	Ok
Send a value of 0x64 (100 decimal) on the VSOME/IP protocol for minimum distance	No change	Ok
Send a value of less than 0x64 (100 in decimal) and more than 0x16 on the VSOME/IP protocol for distance	The first SPI channel will read a 0x00 within less than a second	Ok
Send a value of 0x30 over the VSOME/IP protocol for target speed	No change	Ok

Table 8: Vision controller

Action	Expected outcome	Results (2018-12-11)
Connect and start affected nodes properly	—	—
Hold something with a red colour in front of the camera (Android app, or mobile webpage can be found in the project git repositories)	The vision controller should send the message "1X " within 1 second, where 1 means red, and X is any direction towards the red thing	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a green colour in front of the camera but a bit to the left	The vision controller sends the message "21" within 1 second, where 2 means green, and 1 means left	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a green colour in front of the camera but a bit to the right	The vision controller sends the message "22" within 1 second, where 2 means green, and the second 2 means right	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a green colour in front of the camera but centre in view	The vision controller sends the message "23" within 1 second, where 2 means green, and the 3 means centre	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a red colour in front of the camera but a bit to the left	The vision controller sends the message "11" within 1 second, where 1 means red, and the second 1 means left	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a red colour in front of the camera but a bit to the right	The vision controller sends the message "12" within 1 second, where 1 means red, and the 2 means right	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a red colour in front of the camera but centre in view	The vision controller sends the message "13" within 1 second, where 1 means red, and the 3 means centre	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a yellow colour in front of the camera but a bit to the left	The vision controller sends the message "31" within 1 second, where 1 means red, and the 1 means left	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a yellow colour in front of the camera but a bit to the right	The vision controller sends the message "32" within 1 second, where 1 means red, and the 2 means right	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Hold something with a yellow colour in front of the camera but centre in view	The vision controller sends the message "33" within 1 second, where 1 means red, and the second 3 means centre	Not ok. Timing issues and detection works sometimes. Bad test conditions?
Keep very green, very red, and very yellow coloured object away from the camera field of view	The vision controller sends "00", which represents no control flag object in view	Ok

B Requirements testing protocols

Table 9: Flag following

Action	Expected outcome	Results (2018-12-11)
Start car	Car starts	Ok
Start flag app	App starts and shows a red or green screen	Ok. Yellow is possible as well
Get green flag from the app	App displays a green screen	Ok
Put phone with screen turned towards the car in view of the car camera	Car should move forwards until a distance of minimum 1 meter from the phone is reached, then stop	Not ok. State machine and vision not yet integrated properly.
Move the phone sideways	Car should turn the wheels towards the direction of the phone within a couple of seconds	Not ok. State machine and vision not yet integrated properly.
Move the phone away from the car	Car should move forwards until a distance of minimum of 1 meter from the phone is reached, then stop	Not ok. State machine and vision not yet integrated properly.
Before car reaches the minimum distance, get red flag from the app	Car should stop within 2 seconds	Not ok. State machine and vision not yet integrated properly.

Table 10: Collision avoidance

Action	Expected outcome	Results (2018-12-11)
Start car	Car starts	Ok
Start flag app	App starts and shows a red or green screen	Ok. Yellow is possible as well
Get green flag from the app	App displays a green screen	Ok
Put phone with screen turned towards the car in view of the car camera	Car should move forwards	Not ok. State machine and vision not yet integrated properly.
Before the car reaches a distance of minimum 1 meter from the phone, put an obstacle directly in front of the car	The car should stop before hitting the obstacle	Acceptable. Car stops within a reasonable time. Could only simulate.
Put phone with screen turned towards the car in view of the car camera	Car should move forwards	Not ok. State machine and vision not yet integrated properly.
Before the car reaches a distance of minimum 1 meter from the phone, put an obstacle in front of the car but off-centre to the left or right	The car should stop before hitting the obstacle and turn in the other direction	Not ok. State machine and vision not yet integrated properly.

Table 11: Fault states

Action	Expected outcome	Results (2018-12-11)
Start car	Car starts	Ok
Test operations as above but do one of the following things:	Car operates as normal until fault is injected	—
—	—	—
Disconnect ethernet cable for distance-steering node	Car stops within 2 seconds and stays still until boot up without faulty nodes	Ok. Slightly more than 2 seconds with debug compilation. Could be improved?
Disconnect ethernet cable for motor-speed node	Car stops within 2 seconds and stays still until boot up without faulty nodes	Ok. Slightly more than 2 seconds with debug compilation. Could be improved?
Disconnect ethernet cable for SDN-controller node	Car stops within 2 seconds and stays still until boot up without faulty nodes	Ok. Slightly more than 2 seconds with debug compilation. Could be improved?
Disconnect ethernet cable for vision node	Car stops within 2 seconds and stays still until boot up without faulty nodes	Ok. Slightly more than 2 seconds with debug compilation. Could be improved?
Disconnect ethernet cable for car controller node	Car stops within 2 seconds and stays still until boot up without faulty nodes	Ok. Slightly more than 2 seconds with debug compilation. Could be improved?
Disconnect any SPI pin for speed sensing node	e Car stops within 2 seconds and stays still until boot up without faulty nodes	Not ok. Not implemented
Disconnect any SPI pin for distance sensing node	Car stops within 2 seconds and stays still until boot up without faulty nodes	Not ok. Not implemented