Computing and Management
COC252
B728994

Cost Effective Data Management for Electric Motorsport Teams

by

Carlos Lagoa

Supervisor: Dr. H. Nevisi

Department of Computer Science
Loughborough University

April/June 2020

Contents

List of Figures

List of Tables

1. Abstract

LUMotorsport is a university team that competes in an international engineering competition where a single seat race car is built and raced every year. Sometimes the performance a team attains is due to the monetary funding received and not a reflection of the team's true capabilities, especially in typically non-engineering areas. This effect can be direct (more money, better parts, faster car) or indirect (more money, better tools for the team, better car, faster car).

This problem could become bigger this year as LUMotorsport will transition from a combustion car to an electric one; thus adding not only the yearly build costs, but also the repurchase of 'one-off' items that worked in the combustion car but will not work in the electric one.

This project describes the design and implementation of a low-cost in-house replacement of a series of data-centered technologies which are normally bought from companies as closed-source packages in these 'one-off' purchases. These solutions will save money, allowing other purchases to be made and for improvement down the road, as they are produced in an open-source manner. Furthermore, this will give the team more points in the competition: the judges prioritise in-house solutions over 3rd party store-bought ones.

A communication protocol is be implemented across all the sensors and actuators of the car using the CAN bus standard. One of the network's nodes is a data logger, integral when analyzing and improving both the car and driver back in the garage. In order to utilize the data logger two telemetry visualization systems are to be made: one for critical systems in real-time over WiFi and a more informative multi-platform desktop software that replaces an expensive solution.

Overall, this project achieved its conceptual goal of freeing up budget for other development areas by creating the aforementioned solutions, despite the COVID-19 pandemic limiting the implementation of the hardware projects. The software projects provided an intuitive and responsive user interface for the analysis of the car and driver's performance.

2. Introduction

2.1 Formula Student

Formula Student is an international design competition held in different tracks around Europe over the summer. The competition is used above all as an entry point for engineering students into the automotive industry, and it is viewed by employers (anywhere from a small engineering firm to a Formula One team) as the standard for graduates to meet.

There are 8 events around Europe (and more over the world, represented by other organisations, such as FSAE for America). Teams will typically attend anywhere from 1 to 3 events in one 'season' (which are over summer)

- FS NL, Netherlands
- FSA, Austria
- FS ATA, Italy
- FS EAST, Hungary
- Switzerland, Switzerland
- FSG, Germany
- FS UK, UK
- FS SPAIN, Spain

The way a team can qualify for an event is based on an online quiz which only allows in x amount of teams, typically an event will fold from 50 to 100 teams. The quiz is drawn from the Formula Student Rules, a combination of documents covering above all technical regulations of the car, aggregating to about 140 pages. The team's that answer the most right questions in the time given are the ones to attend the event.

2.2 LUMotorsport

LUMotorsport is Loughborough's University Formula Student team, founded in 2003. It produces a car to compete in the Formula Student competition every academic year.

It is composed of around 20 students, with around 40% of those students being final year students. Out of the remaining 60% half will be in other 'Senior Positions' (ie, have been in

the team for one academic year or longer) and the other half will be first year team members (typically 'freshers').

The team members are then further split into different subareas that regard the development of the car as well as the management of the team. Some team members might choose to work on designing areas of the car, some might choose to work on the physical production of the car and some might choose to take a more managerial role as team leader. These roles are established via an informal chat before the new year starts.

This splits allows the team to always have a steady flow of members dedicated to different areas of both self improvement and overall team improvement.

For example;

- Team Member A might be a second year student that has done a placement. He spent his first academic year as a 'fresher' working on the build process of the car and learning the processes that make up the chassis design. He then went to placement to a firm where he learned more about the Computer Aided Design (CAD) involved in the chassis build. And then on his third academic year, he will become to Chassis Head, where he will have to be in charge of designing that specific part.
- Team Member B might've moved around a bit more than Team Member A, going from one department to another one, however this gave him the understanding to see how the whole team works together and what department might cause delays, or additional stresses. He might want to become to Team Leader for his final year and work on the the communication and transparency of departments.

It is this steady flow and constant improvement that have made LUMotorsport one of the UK's best teams, ranking an impressive 4th (out of around 100) in the FS UK event last year.

I joined LUMotorsport mid way through my first year and became the first student to not be enrolled in an engineering-based course.

This presented some issues as I wasn't as knowledgeable in some areas as the other 'fresher' team members - who typically work on the build process to get an understanding of the car's aspects. Instead, I was teamed-up with the Electronics Head that year who was trying to implement a new steering wheel display, as an improvement from one the team had used for a couple of years. This was how I got interested in the Data Management areas of the car.

I spent my second year doing work on other parts of the competition that weren't related to the car, and for my third and final year (and Formula Student season) I have been part of the

Electronics Team and Driver Controls, who I have worked with in order to make the switch from a combustion engine to an electric motor as swift and cost-effective as possible.

2.3 The competition

The Formula Student competitions are organized around the car's performance and the team's management of the car's build.

Table 2.1: Formula Student Event Points [1]

Event Type	Points
Static Events	475 points
Dynamic Events	525 points
Total	**1000 points**

In short, when a car is fast it will perform better in Dynamic Events and it will get more marks from that area.

When a car has been produced using new techniques to make it lighter, cheaper, etc... etc..., or when a car's itemized budget is more accurate, or when the car's techinical features are presented better in front of competition judges, it will get more points from the Statics Area.

2.4 Problem Definition and Motivations

For this current academic year (2019-2020), LUMotorsport has decided to shift to an electric motor. This presents changes to almost every part of the car;

- Suspension Geometry needs to be adjusted for the new center of gravity from the batteries
- The chassis needs to change to account for the dimension's change.
- A large part of the electrical sytems need to be reworked.
- and more

However it is this final area where we will focus our attention. Currently the team employ a combination of tools (both software and hardware) from MoTeC - a company specialising in engine management, data acquisition & display and power management systems for a wide range of motorsport areas and disciplines.

2.4.1 Problem 1: Cost and current solutions

The problem with MoTeC all along has been its cost, as for several years they were the only company providing certain solutions and they have stayed with these premium pricing. For bigger motorosport teams MoTeC can prove to be indespensible, as it can perform tasks some other third-party solutions can't implement.

MoTeC have created an ecosystem for their solutions, which means that (perhaps similarly to Apple products) the best experience will come to the end user when more than one piece of hardware or software work together. Additionally this means that even if a part is cheap, the size of the parts bought will increase - which makes the cost of the system increase too.

In general terms and for the concern of this project, the way LUMotorsport has been using MoTeC parts has been in the following way;

The car's sensors will produce an output, typically at anything between 20Hz and 100Hz.

- MoTeC will collect the 'SensorsA''s readings directly through 'Network A' as there is another another MoTeC product in the engine that isn't described in this diagram (due to it not directly influencing this project). This data will be passed to a MoTeC data logger in the car, and then when the car makes it back to the pits, the team will download the data logger's readings into a team laptop, which runs a software by MoTeC which will convert the MoTeC specific data log into a chart visualizations. - For 'SensorsB', 'SensorsC', 'SensorsD' and 'SensorsE' the process is very similar to the one described above, only that 'NetworkB' is made by the team as the sensors in the other areas of the car need to be more flexible than the engine ones.

As one can see after reading this, it is fairly easy to see the monopoly that MoTeC has over the car. And that adds to the cost.

- A MoTeC data logger costs around £900.
- The engine management software that makes up the connection through 'Network A' to 'Storage' is roughly £2,000.

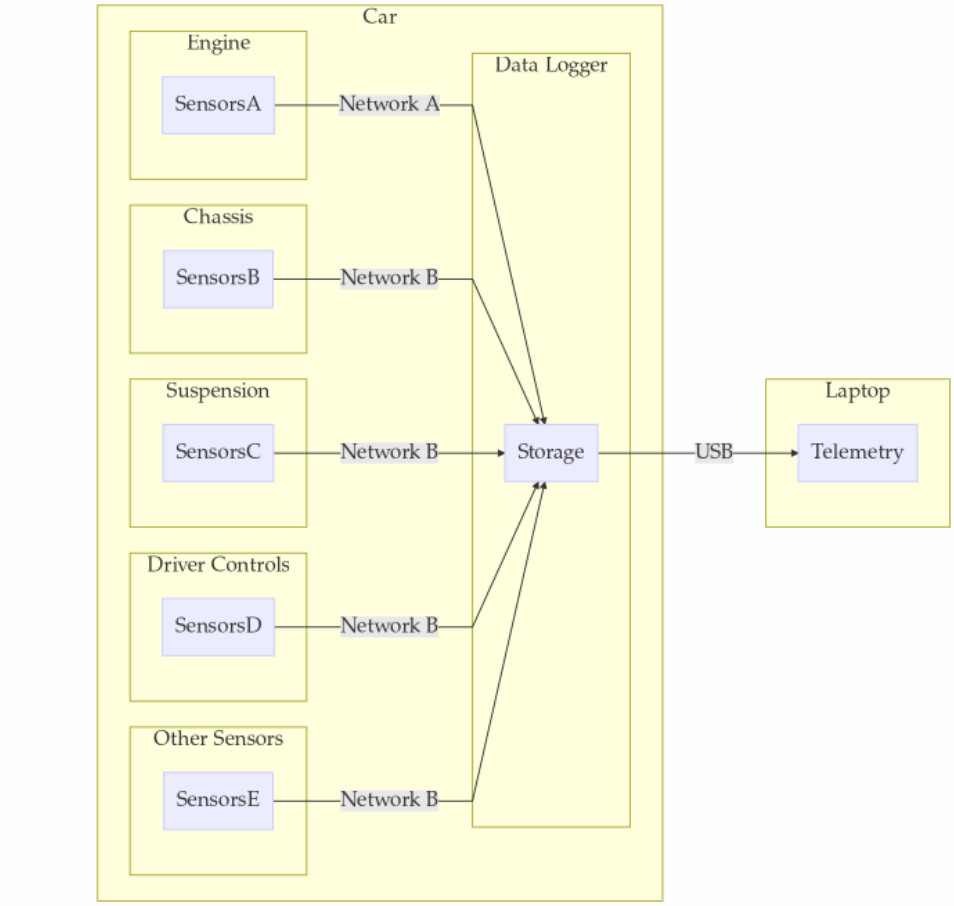


Figure 2.1: Car's current setup

- Although the telemetry software is free, that is because it is being rendered from a MoTeC-specific log file (thanks to the Data Logger). If one wanted to import a custom .CSV of the data that would be an additional £1,200 per year.

- The telemetry software itself isn't upgradable, nor does it have all the functionality it could have. If a team wanted the proper 'fully featured' software that would be a £2,700 one time payment, following by a recurring yearly £500 'subscription'. This allows a team to enjoy the full version of the Telemetry Software, without any sort of live telemetry and without any sort of third-party accessories. If those are required, for example a GPS module or the ability to import a dataset from a 3rd party company, that would be additional thousands.

2.4.2 Problem 2: Going electric

As of this year the car, LFS20, will be electric and this means that the primary reason to use all-MoTeC is gone. Reading Fig 2 backwards its easy to see the decision process for using MoTeC technologies;

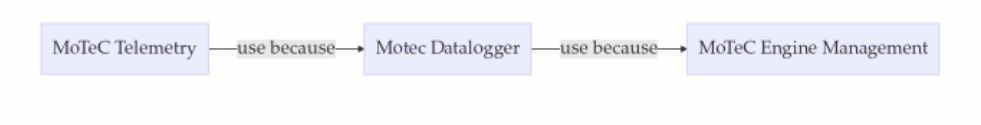


Figure 2.2: MoTeC's justification

An electric motor isn't a combustion engine and therefore all of the MoTeC specific engine sensors and parameters will not be used again - this means that the team will have to make their own electric motor sensors or purchase them from a third party. If we do not rely on the Engine management system it is a lot easier to deal with a custom made data logger, since we will be able to have a direct say over every sensor in the car.

By extension, if the data logger can be made by someone else that isn't MoTeC, then why pay the MoTeC £1,200 'Custom Dataset Import' Fee for the software?

All of this questions have an answer, the answer is to move forward as a team and step up to the challenge of custom building a replacement for every MoTeC solution ourselves. Half doing it won't do, as failing at any stage of the progress would incur in the next segment of the process picking up extra charges.

For example, if a data logger can be made custom but the software can't then we would have to pay MoTeC's to read our dataset. Or, if we can implement everything but the Engine Management System's , which from this point onward becomes known as the BMS's (Battery Management System), Network System then money would have to be spent on the a data logger which will adhere to the specifications of the Network topology implemented.

2.5 Aims and Objectives

2.5.1 The influence of COVID-19 within this project

Because of the virus that has stopped the world for a couple of months, some of the parts detailed in this project have had to stopped and therefore some might only be available as a

'proof-of-concept' rather than a fully integrated system. The parts that have been affected by this will be labelled as such.

It is perhaps difficult to see how the influence of a lockdown could inhibit the progress of a Computer Science project. However, bare in mind that this isn't an individual and only-software project. This is a team effort, where I control the software based activities but a team effort nonetheless. This means that if the workshop where the car is built is closed, I will not be able to access some parts to code, nor will I be able to communicate with the other departments regarding their parts, as they might have shifted their attention to something else.

The reason for this attention shift is because this year's Formula Student events have been cancelled and therefore teams now have a 2 academic year gap to perform changes on the car. This has greatly affected the initial layout and timings of the project, and has made it impossible to deliver upon some of the objectives.

2.5.2 Aims

The aims of this project will be a subset of smaller problems to solve. All within the environment of data and its implementation on the new car - in order to achieve the team's objective of becoming completely independent of the MoTec suite, for the reasons outlined in section 2.4 above.

This project will (even without COVID's influence) need some final tweaking after this report is written, as the Formula Student scene is no stranger to last minute changes, either based on new requirements or regulations. Nonetheless, almost all aspects of this project have been developed to an extend where they are more than a 'prototype'.

On a personal note, I hope to increase my knowledge and better my approach to the following topics;

- Technical Knowledge; I hope to increase my knowledge of implementing cross-platform applications using HTML, CSS, Javascript, using NPM (Node Package Manager), NodeJS and RStudio's Shiny Server. I also hope to learn about Arduino and C, and to get a basic understanding on the basic electronics of a car's network, using the CAN bus.
- Team working; After two years in the team this third year seems like the perfect opportunity to step up and become the head of a team, this means that the responsibility of delivering a part of the car's fundamentals is on my plate. This will not only test my critical thinking abilities, but also my communication skills.

The project will be split into;

- Creation of a communication network using the CAN standard
- Creation of a Data Logger
- Live Telemetry
- Data Logged Telemetry Software

2.5.3 FAST Goals

FAST goals will provide me with an alternative to the classic SMART (Specific, Measurable, Achievable, Relevant and Time-bound). The problem with SMART goals can be that they focus on individuality rather than team communication and that they do not provide the ability to provide flexibility [2]. As an alternative, FAST goals focus on Frequently discussing the areas of work, setting Ambitious (rather than achievable) goals to encourage the work to be pushed a bit more, Specific (as they are with SMART) and Transparent which allows the flexibility a project this size might need and the communication and coordination that the departments involved might need. FAST goals can be found below.

2.5.4 Risks

The risks associated with this project relate mostly to time constraints and opportunity cost of using 3rd party systems. Regarding the time constraints it is important to primarily with the hardware solutions, as its development needs to be finished and implemented into a skeleton-like chassis of the car so that the Electronics department can understand the placement and power requirements of each component. This has to be done before a scheduled deadline based on the feedback and progress of the Chassis, Suspension and Manufacturing teams.

In regards to the telemetry solution, the risk of non-completion comes primarily from an opportunity cost point of view. Opportunity cost relates to the second-best alternative foregone, in this case the second-best alternative is a 3rd party solution and the chosen alternative is this project's implementation of the solution. If mid way through the project it is concluded that the requirements cannot be met it would mean that it would be too late to change the situation and therefore the team might have to spend the next year with a very simple and ineffective visualization system for telemetry; for example loading the data into Excel.

2.6 Outline

Find below an outline, including a brief description of each chapter of this report. Each chapter aims to provide a different dimension to the completion of the project, from discussing alternatives to the final presentable system.

Table 2.2: FAST Goals

Goal	Discussed Freq	Ambitious	Specific	Transparent
Create communication network	Weekly meetings with Electronics Department	Implement it using the industry-standard CAN, rather than Serial communication	Take all sensor readings from car	Reporting once a fortnight in team meeting
Create data logger	Weekly meetings with Electronics Department	Implement a solution smaller than the current MoTeC data logger	Takes CAN and outputs a .CSV file	Reporting once a fortnight in team meeting
Live telemetry stream to laptop	Weekly meetings with Driver Controls Department	Implement something that currently MoTeC does not supply us with	Graph-based, not just text-based	Reporting once a fortnight in team meeting
Telemetry Visualization Tool	Weekly meetings with Technical Chief, as this projects falls completely within my department without the need for input from any other department	Implement the solution using open-source software that can later be scaled to meet the team's future needs	Must adhere to decided specification	fortnightly Demos to key team members

Table 2.3: Report Outline

Topic	Subtopic(s)	Summary
Introduction		Introductions made to the physical environment, ie, the client of the project, as well as the problems their current solution(s) have.
Literature Review & Background	Communication Protocols	Aims to explain and describe the two common solutions that teams implement.
	Telemetry Solutions	General information about telemetry tools with examples.
Specification & Methodology	Hardware-based projects	Describing the hardware project's scope, requirements and development methodologies.
	Software-based projects	Describing the hardware project's scope, requirements and development methodologies.
Project Management		The Gantt chart followed as the main outline of the project, as well as other sprint-based management systems for smaller tasks.
CAN Network	Design, Implementation & Testing	Outlining the increments to develop the CAN network.
Data Logger	Design, Implementation & Testing	How a smaller footprint data logger was custom made for the outputs of the CAN network.
Live Telemetry	Design, Implementation & Testing	A look at the options considered and the basic implementation of an upgradable package
Telemetry Software	Design, Implementation & Testing	Description of the software-based project and its iterations
Results		A direct comparison between the project's specification(s) and the work produced.
Conclusion		An overview of the project's wins and losses, as well as possible further developments.

3. Literature Review & Background

3.1 Communication Protocols

3.1.1 Their place in motorsport

Any mechanical system, with the focus of an automototive team for us, will typically have similar requirements regarding the data management. There will be a large number of sensors which at any might need to;

- Be used as the input for actuators; for example, in new generation super cars a rear wing will be extended depending on a combination of factors such as current brake pressure, vehicle speed or engine setting among many others in order to increase or decrease the car's wind resistance.
- Be displayed; critical driving information must be displayed to the driver in real time in order to ensure he stays engaged with his environment an example of this is the gear position or fuel level.
- Be stored; just because sensor information is not time critical does not that mean that it it should be 'chucked away'. In scenarios like testing or car setup it is incredibly beneficial to have as much information as possible, for example suspension information.
- Be used as security device; Another example of real time data processing can be in order to make a decision faster than a human could. In any motorsport competition sometimes the engine will shut off, and more times than one might think this is due to a self-protection mechanism rather than an 'actual blow out'.

With this list we can start to see the importance of sensors in motorsport. Perhaps LUMotorsport does not require a sensor layout as complex as a F1 or Rally team, but nonetheless even a relatively 'small' project like ours requires an array of sensors that will 'pile up' if not carefully laid out.

In the following paragraph we will look at the main different ways to create said networks, and at their advantages and disadvantages.

3.1.2 SPI Protocol

Commonly known as just Serial, the Serial Peripheral Interface is a synchronous serial communication interface, which means that it has it has one source for the timing of a network.

SPI counts with a Tx (Transmitting) line, a Rx (Receiving) line and (as specified above) a clock line. The clock is driven by one side of the interface, which is called the Master. The master controls the clock, and therefore the master controls the flow. This can become useful to avoid interrupts and live-data errors. However depending on the slave node (the opposite of the Master node) this can become problematic. [3]

If the slave node is a simple sensor then it will easily always send a value back when requested by the Master. However if the slave is a more complex system then that might generate a delay for the Master, waiting for a response. This can grow to be a big problem if more and more slave nodes are added to the network.

This is because SPI communication will only connect to one slave at a time, and it will wait for that slave to communicate back [4]. In fact, sometimes when the Master is sending a message requiring no response from a Slave, it is commmon place for the Slave to send a 'dummy message' back [5] in order to 'unhook' the Master and let the flow of the system continue. This type of communication can also be a problem if one of the slaves becomes what is known as a Bad Actor, whereby it sends wrong information that may never close. For a team that is made of volunteering students using third party sensors, debugging a problem with a Bad Actor can be quite the challenge.

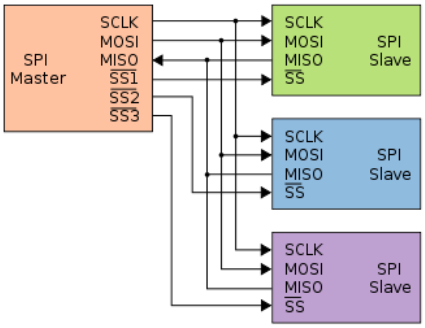


Figure 3.1: SPI Layout [6]

As we can see in Figure 3.1 the Master has;

- 'SCLK' which is the Serial Clock to all Slaves connected
- 'MOSI', Master Out Slave In, where the data for the slave will go.
- 'MISO', Master In Slave Out, where the data from the slave will go, always to the master.
- 'SSX', Slave Select (sometimes called Chip Select), which is the line that will individually 'select' the Slave it wants to engage in communication with.

3.1.3 CAN Protocol

Another alternative to SPI which could be seen as the most 'professional' approach to data management in automotive settings would be the Controlled Area Network Protocol. Developed by engineers at Bosch as they couldn't find a protocol which fit the standard they wanted to implement [8].

From an abstract level, one of the main reasons CAN is seen as superior is because of its standardisation within the automotive and automation environments.

Every node connected to a CAN network will always send the same data layout [3]. This same data layout allows for packets of data to be sent instead of single bits one after the other (as is the case with SPI), the standardized packet is implemented as a hardware feature on CAN.

Therefore a node can place a message inside a packet, the message will be joined by a checksum (a set of numbers and digits that relates to the length of the package sent, which is then compared against the checksum that the receiving node got [9]) among other items defined below, and not worry about having to 'interact' with the data until the packet is sent or received.

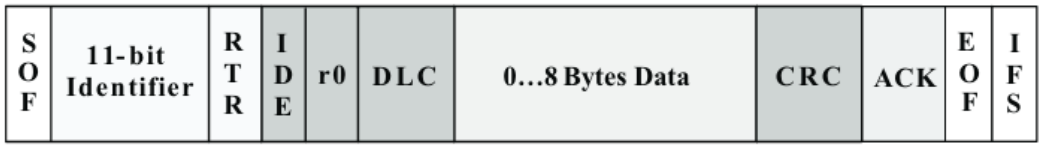


Figure 3.2: CAN Packet [11]

The contents inside a CAN packet are;

- 'SOF', this 1-bit identifier defines the Start Of Frame.
- The identifier establishes the priority of the message and the unique indentifier.
- 'RTR', Single Remote Transmission Request is 0 when information is required from another specific node. All of the nodes will receive the request but only the one specified in the Identifier will get the message.
- 'IDE', Identifier Extension bit will used when responding to a request from another node.
- 'r0' is reserved for possible future use.
- 'DLC' is the Data Length Code and it contains the number of bytes of data being sent.

- 'Data' defines the being sent, which may be up to 64bits of data (8 bytes). If less than that, the data must still be filled with 'dummy values', similar to the 'dummy responses' sent by Slave SPI nodes.
- 'CRC' is the Cyclic Redundancy Check and it contains the checksum for the package sent previously.
- 'ACK' is the Acknowledgement of a well received package, if 0 the data is okay. If 1 (which is the default value the package arrives with) the data will be resent. This is a mechanism called Backoff-and-Retry is similar to the ones found in technologies such as Ethernet [10], whereby the system will decrease the rate of a process in order to allow it to 'catch up'.
- 'EOF' is the End Of Frame, with some additional mechanisms such as Bit Stuffing, which ensures the integrity of the package by filling any 'unfilled' frames.
- 'IFS', is the Interframe space, and it allows for time in order for the receiving node to change the parameters of the message.

3.1.4 SPI vs CAN; Topologies

After reviewing the basics concepts of the two methods, we can put them in context with an example. This will allow us to see one of the three main benefits of why CAN might be considered better than SPI in our scenario. We could think of a SPI communication network in terms of topologies as [7].

As seen in Figure 3.1, the SPI layout presents us with a Star Topology, whereby [13] even though circuit control is more approachable - because it all passes through the main switch, in our case called the Master.

However there are is, comparatively, a bigger disadvantage with this topology and that is the fact that if the Slave fails the whole system will fail. This is an incredible risk to take, and a risk which turns into failure with a higher than expected turnover ratio. Again, after all, this is a student based event, where students will design the car and its supporting structures as well as race it. This translates as a general design rule to: "whenever it is possible to design contingency failure system, do so".

In contrast, a CAN network presents itself as a Bus Topology. A topology of this kind still presents itself with a single point of failure [14], this denotes a system where a single manfunction can bring down the entire network.

However this point of failure regards the 'communicating cable' rather than the nodes [13]. This is what a team, at least a team like LUMotorsport will be looking for, as it is much more likely for a node to break due to its connectivity with other systems than the actual connecting cable.

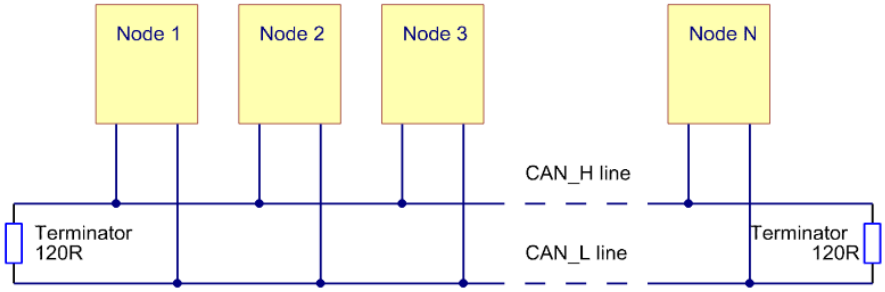


Figure 3.3: CAN Topology [12]

In Figure 3.3 above we can see the CAN bus topology. One final piece of information regarding CAN's topology is the 'CAN_H' and 'CAN_L' lines - CAN High and Low. When the bus is in an idle position (no data being sent) both lines carry 2.5V [15]. However when bits are being sent along the line, CAN High goes up to 3.75V and CAN Low goes down to 1.25V - thus generating 2.5V differential between the lines. This makes the bus resistant to interference from noise such as electrical fields or inductive spikes, something extremely useful when in a rattling and shake-prone environment.

3.1.5 SPI vs CAN; Node complexity

The second main advantage regarding CAN over SPI concerns the complexity of the systems in the car that will be part of the network. Previously MoTeC has taken care of the most complex part of the network; the engine. MoTeC's ECU (Engine Management System) is directly linked to the MoTeC data logger.

This means that everything LUMotorsport is tasked with doing is getting other sensors to 'report' to the data logger over a SPI network. Things like that suspension readings are extremely useful however not essential to the safe running of the car.

This directly translated to LUMotorsport implementing the 'easiest' solution (which we have discovered in this past chapters to be SPI), as it was time effective in both research and implementation. However, change is something inevitable, and as such, now the team have the duty to take sensor readings from what would once be part of MoTeC's integrated network.

One important final part to mention regarding SPI and CAN is that whilst SPI is meant to

connect integrated circuits together, CAN takes this a step further. CAN is 'out-of-the-box' ready to deal with the inter connectivity of complete modules and it does so with better security and reliability functions than SPI does [4]. As we mentioned before, one of the main advantages of CAN (which translates as one of SPI's main disadvantages) is the standardization of the protocol.

This can be seen even in the philosophy of the CAN network, whereby one of the parts of a packet is simply left for future upgrades. It can also be seen in the market and community approach to CAN; with updates [8] to the protocol at key points and a wide range of accessories. This doesn't meant that SPI does not have accessories or support - at the end of the day, most of the Arduino and other hobbyist micro-controller community is built on SPI, however it does mean that the CAN community is more orientated towards larger engineering projects such as the one LUMotorsport is involved with.

3.1.6 SPI vs CAN; Formula Student

This last part is non technical, and it references Table 2.1 'Formula Student Event Points' and the way points are distributed. There is a large part of the points, around 10% of the whole competition which will be based upon a Design Report presented in front of the judges.

This presentation will feature the judges seeing every design decision made on the car that has been made as a development as part of last year, and then the judges will ask questions regarding the justification for those decision [1]. As a team, LUMotorsport would benefit from implementing a CAN network in the following ways;

- Lower cost of implementation; as a custom CAN network would beat the cost of a pre-made solution such as the one provided by MoTeC.
- Better understanding of the car; the judges will reward teams that make problems their own and solve them using their understanding of the core concepts of that problem.
- Improvement over a past version; this means that the team is passionate about not only the first generation of their projects, but also on working on improving them. A clear benefit in the world of engineering.
- Adaptation to the new environment; as LUMotorsport is going electric this means that many changes will need to be made, the judges encourage teams that made early decisions and work on a complex implementation rather than trying to stick to old systems on the final minute.

- System as whole; coming back to the fact that this is a student based event - the judges will be proud to see a team implementing a solution which is a standard in the automotive industry.

3.2 Data Loggers

Data loggers are electronic instruments which register data inputs from incoming channels at time intervals. A data logger is typically generic, at least compared to its implementation. Data Logger A and B might be produced by the same factory however one might end up being used to register a garden's properties (air temperature, humidity, light intensity...) and another might be used by a research team to log the path of animals across different smells (time taken to interact with the smell, animal pace...).

In a sense they are like a printer or a canvas, whereby the allow the user to customize their use based on the inputs they have at hand. Much like a printer and canvas, data loggers also tend to be relatively low cost (especially for non-specialised areas such as the examples described above), easy to use and customizable (with the end user ranging from a secondary student to space uses) and reliable - as once configured they do not need to be removed from their location, thanks to the fact that the device that carries that actual data tends to be an SD card or a wireless technology. [16]

3.2.1 Main components

The main considerations when implementing a solution regarding data loggers are [17];

- Input Channels; channels relate to the physical connection between the sensor's output and the connection to the data logger's data processor. A simple data logger will have a single sensor outputting to a single channel. However it is possible for more complex solutions to count with multiple channels in order to log multiple data. It is also possible for a data logger to output a channel with multiple inputs, this would be achieved by the logger's data processing capabilities whereby inputs are processed and presented as one.

  The channel types can be;

  - Analog Channels, the most common, output a voltage based on the value they work against. For example, the Arduino community will often use things like light sensors, accelerometers and pressure sensors [19].

– Digital Channels describes the values being outputted as discrete, this is 1s and 0s, or a step-like plot if graphically represented. The reason for a sensor to be digital is presented when the operations carried on are more complex, an example of this could be a digital signal being carried to an integrated circuit with a microcontroller which treats the data. This aids in scalability, as it prevents from a main 'hub' from treating all the analogue data at once. [20]

An interesting fact, that also aids in the visualization of digital and analog signals comes from the SONY VAIO logo [21]. Whereby both signal types are shown as part of the logo design.

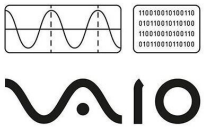


Figure 3.4: VAIO's logo

- Analog to Digital Converter; All sensor types must enter the data logger's data processing unit in Binary.

  Binary is a base 2 numbering system, rather than our conventional base 10. The way we read binary is by adding up all of the indexes recorded as 1s rather than 0s. The indexes are made of their position to the power of 2. Below is an example of two readings entering the data in 8-bit (8 indexes) of binary;

Table 3.1: Reading binary

0^2	1^2	2^2	3^2	4^2	5^2	6^2	7^2	Addition	Total
0	1	0	0	1	1	0	1	$1^2 + 4^2 + 5^2 + 7^2 =$	77
1	1	1	0	0	0	1	0	$0^2 + 1^2 + 2^2 + 6^2 =$	226

  This method is simple for digital sensors as the values already come counted, however for analog signals the story can be a bit more complex. They will need an Analog to Digital Converter; these take the samples sent from the analog signal per second (known as Sampling Rate) and the resolution of the data. The resolution will show the steps to make per varying voltage, and the frequency will tell how often to make these calculations [22]. Once this is achieved the data logger will receive a digital value.

- A micro-processor will also be integral to the functioning of the data logger. In short, it will be where the data logger 'understands' the processing that is needed to be made for the incoming signals and how these digital signals should be outputted to the next stage (the data logger's storage).

It is normal for the micro-processor to be a modular part of the data logger, something like an Arduino will work well enough for most of the cases.

- A data logger will need a memory for the loading of its instruction set to the micro-processor. This can typically be achieved by two ways (depending on the environment its power supplying capabilities). One option is a volatile source that is constantly loaded from another bigger system once powered on (such as RAM) - this would work well for systems that are stored in a workshop and aim to be minimal and used for research or prototyping, as they will have the ability to always be wired to a computer that is ready to change the data collection process.

  The second option is a non-volatile solution which then gets loaded to the chip's RAM once power is resupplied. This is the most popular case, as it also allows for quick modification, and along the years it has become the industry standard with solutions like Arduino using this method.

  Apart from needing memory to load from, a data logger will also need memory to write to. The most common approach, and the one that takes advantage of the modularity of such 'basic' electronic components is using an SD card or any other type of flash-like memory which can be then removed an read in another environment.

- As explained above a large part of the memory selection process regards the power supplying capabilities of the environment the data logger is in. This is quite a straight forward decision. For a data logger that collects garden information it will make sense for it to be plugged in to the house's mains connection. However for other environments, like a moving Formula Student car, or any moving car for that matter, it will only be possible to have the data logger get its power from a battery.

- Enclosure is the last part when considering the importance of choosing the right data logging solution. One of the main sources for the failure of a data logger will be environmental issues [23] and it is therefore essential to keep the logger in an enclosure that protects it from the harsh conditions of its environment. In Formula student this translates to water-proofing and shake-proofing the logger. In smaller environments there might also be a need to protect each circuit from interfering with each other.

3.2.2 Their importance in motorsport

Apart from the importance a data logger might have to every kind of project there is a another reason why a data logger will be indispensable in motorsport.

As mentioned in the section above, a data logger will serve as a 'hub' for all the sensor data which has already been curated (either by adding channels together or generating the sensor

inputs as binary). This means that other devices apart from a memory card can take advantage of this data. This ranges from 'luxury' items such as lap time counters based on GPS signal or live telemetry to the pit wall, but also with something far more basic and necessary.

A data logger will produce output to a car's dashboard [18], this will give drivers information they need in order to drive the car to its limits.

Figure 3.5: Motorsport Dashboard

3.3 Telemetry

The word telemetry refers to the process of transmitting data measurements from one location to another one. This process is, like a data logger, a printer or a canvas - an initially 'simple' process which can then be tuned and tweaked for the end-users goal. Literally, telemetry can be the instrument readings of a plane, the ink left on a printer and even the time left on a microwave timer.

However, for this project telemetry is approached as the visualization of this transmitted data rather than the process of transmission as such. Data visualization technologies have gained considerable traction since computers has become cost effective, simple and small enough for teams to use whilst 'on the road' and now more than ever there are thousands of platforms and graphs for these visualizations to happen [29].

Another aspect which has helped the growth of telemetry and has gone hand in hand with the introduction of smaller and faster computers is data science, the "sexiest job of the 21st century" [30]. It has allowed more data to be processed, this helps teams in ways such as analysing trends.

3.3.1 Reasons to use it

There are plenty of reasons for teams to want to use telemetry, especially if the software that displays the data is dynamic and easy to use. Here are some of the reasons why telemetry has become essential to even smaller teams such as LUMotorsport [28].

- Analyse vehicle performance; by checking the many readings a car will produce around a lap it becomes easier and more intuitive to understand the driver's handling opinions, and to be able to detect key areas where the car might perform above or below average.

- Analyse driver performance; this aspect allows the driver to be inspected, in relation to other laps (in order to find the best approach to parts of the circuit) and to other drivers (in order to establish who the quickest driver is). This data can then be checked by a track side engineer and used as feedback for a driver's next run. This type of performance is the one which can be most rapidly modified [27]. It is also important to bare in mind that some driving styles match a car and others match another type or car - information like this is also useful when developing a new model.

- Help with development; apart from the help received from a car's response to certain drivers and driving skills, 'pure data' (ie, without any need to analyse anything on top of it such as drivers) can also give the teams an advantage. Figures like fuel efficiency, tyre wear, aerodynamic resistance are concepts which telemetry helps understand.

- Safety and reliability monitoring; recording critical parameters like tyre pressure, oil temperature and battery voltage can help test the reliability of the different parts of a vehicle, as well as help understand what went wrong if anything does go wrong.

- Recording of parameters; this area might only classify for teams with a higher budget to make use of the data collected. However, it is well known that racing series such as F1, Endurance Championship and NASCAR use their own simulators with their own models. Therefore, collecting all these parameters can help when designing realistic simulations or prediction software.

- Engineering back logging will also be helped by telemetry as it will record the strain and timed use of certain vehicle parts, thus letting know the engineers when something needs replacing.

3.3.2 Parameters monitored

Obviously, the parameters that can be visualized from a car's sensors are plenty, however here are some examples of readings that a team would be expected to always have [25] [31];

- Braking - typically in the form of pressure on the pedal.

- Wheel speed - this data can often be recorded as part of a 4 channel network (front left and right and rear left and right tyres) and might allow the team to get more insightful information regarding how much more tyre A moves compared to tyre B along a corner.

- G Forces - these where one of the first telemetry readings to be picked up by TV broadcasters in the Formula 1 series, it shows the lateral and longitudinal forces a car will experience.

- Engine readings - these were done by MoTeC for LUMotorsport, and returned a wide variety of sensor readings such as engine oil temperature, fuel pressure, manifold air pressure, etc etc.

The list carries, and even for smaller teams such as LUMotorsport the list does indeed carry on for a much longer stretch. The latest iteration of the the car, for the 2018-2019 season, reads 104 individual channels of data which are then viewed as telemetry - though some of these are error logs so not necessarily informative at all times.

3.3.3 Real time

Live telemetry is also an option which teams with a big capital very often, if not always, implement. Slowly we are starting to see smaller teams implement bespoke solutions from what they consider fits their requirements. Some teams will use the 3G network [32], others radio and others WiFi.

For these smaller teams it is often only the most essential data which is sent, as when it comes to wireless communication sending more data is indirectly proportional to the lenght that data can carry.

4. Specification & Methodology

4.1 Specification

4.1.1 CAN Network

This document was decided in conjunction with key players in the development of LUMotorsport's LFS20 electric vehicle, who are my clients. It concerns the development of a CAN based communication protocol as the main network to transfer information between sensors, storage devices and actuators as the replacement for the solution used in the years prior; a proprietary MoTeC-based package for engine management and an in-house built Serial-based communication network for every other part of the car.

This project prioritises quick prototyping and good communication with other departments, as the project itself may turn in a new direction at a moment's notice. This is because of the way the Formula Student development process happens and the added volatile nature of this year's powertrain change, from combustion to electric.

The way the specification was decided was slightly different than the 'traditional' approach to software development. This is because the Electronics department was yet finalizing a layout of the sensors of the car and the High Voltage Department (in charge of the electric motor, accumulator, batteries and battery management system) was having to redo a section of the battery layout and then had to file a Rule Clarification Query with the Formula Student organizing body.

The result of this was that the CAN network's beginning would be vague and that, as other departments did more work, more data would be fed into the development of the CAN network. After a meeting at the beginning of the academic year it was clear that there were other priorities for the first two months of development and that after that work on the CAN network could begin, yet pending to see how many sensors it would need to transfer data to.

Therefore the specification was split into two parts;

- Firstly I would gather some working prototypes of the CAN network, advancing my knowledge in its working. This would start around Semester 1 Week 4 and end in Semester 2 Week 4, which provided a lot of extra time.
- Then once the High Voltage and Electronic teams had progressed to make a formal requirements document, I would start working on that. This was expected to start in Semester 2 Week 4 and last until essentially Semester 2 Week 16.

4.1.2 Data Logger

This document was decided in conjunction with key players in the development of LUMotorsport's LFS20 electric vehicle, who are my clients. It concerns the development of an in-house data logger as the next development addition, in the form of a new node, to the CAN network.

Similarly to the CAN Network, this project would be split into 2 stages - with the first one being an almost 'proof-of-concept' stage where I can investigate the ways of integrating a data logger to the CAN network, and the second part being a continuation of my development but with new information in the form of further requirements from other departments.

However, on the contrary to the CAN Network requirements, I was able to discuss a set of requirements with the Technical Lead for this project which where based upon the general requirements and considerations of a data logger, as outlined in Chapter 3.2.1. Again, this requirements are not in depth and are in fact very informal, once again allowed by my closeness with the team and understanding of the requirements. They can be considered of an exploratory and/or prototyping nature.

1. Treating digital channels
2. Treating analog channels and ADC creation
3. Micro-processor to treat incoming data
4. Memory to load
5. Storage to offload data
6. Power supply

4.1.3 Live telemetry

This document was decided in conjunction with key players in the development of LUMotorsport's LFS20 electric vehicle, who are my clients. It concerns the creation of a proof-of-concept model which will allow for virtually free of charge live telemetry. When this task was given to me I was especifically told that this was the area of least concern for the team and that if I should focus on something it should be the implementation of the 3 other projects outlined in this report.

Nonetheless, I was given a set of criteria to try to research and a small time to do it - as the team didn't want to focus too highly on this area, and leave it to be a project for someone else

next year. This is because there is no point implementing a semi-functioning live telemetry, as neither the team nor the Formula Student judges will enjoy using it. Therefore the task was to simply "find a couple of ways of streaming critical live data from a small number of sensors to a laptop or phone, so that when next year starts there is at least some orientation".

However the one challenging request was to make it for as little as possible.

4.1.4 Telemetry Software

This document was decided in conjunction with key players in the development of LUMotorsport's LFS20 electric vehicle, who are my clients. It concerns the development of a replacement for MoTeC's telemetry software which is the one that has been used up to now.

The priorities of developing this piece of software concentrate around making sure it does basic functions properly and that it is upgradable as well as documented, this is because the car is in an era of transition and LUMotorsport understands that what is useful today, may not be tomorrow.

The software's requirements also take inspiration from Chapter 2 of "Analysis Techniques for Racecar Data Aquisition" [31], which has been used in the meetings to overview the requirements and for the parties to understand each other.

1. Data Importation;
    (a) The software must accept a .CSV file
2. Data Preparation;
    (a) The dataset presented may have blank readings, the charting software shall still chart those readings.
    (b) Anything up to 40 minutes of data may be passed into the graphing software, the software must be able to handle that.
    (c) Sensors must be separated into different categories, so that these can be plotted in different areas.
3. Data Visualization;
    (a) Once loaded the data must be responsive to a graph's area interaction, ie zooming, window resizing, panning.
    (b) Time-series visualized data;

i. The user must be able to zoom and pan into the graph.
ii. There must be multi-channel graphs covering the Y-axis, all along the same time series
iii. The user must be able to overlap data from different times.
iv. The cursor must act as a pointing device for the data, whereby the data point is highlighted.
v. The user must be able to add a 'point' in the data to reference later.
vi. The user must be able to filter out any overlapping data.

(c) Non time-series visualized data;

i. The user must be able to plot any two channels against each other without the need for time to take part in the plotting.

(d) Map and lap splits;

i. The software must find the point when the car starts a new lap.
ii. Laps must be plotted on a table which shows;
    A. Lap number, including the Out Lap.
    B. Total run time.
    C. Lap time.
iii. There must be a map of the circuit.
iv. The user must be able to hover the data and see the car's position in the circuit.

(e) Gauges;

i. There must be gauges for;
    A. Throttle Input.
    B. Brake Pressure.
    C. Steering Angle.
    D. Vehicle Speed.
ii. The gauges must, like the map, change accordingly to the user's current hover position over the graphs.

(f) Data Playback;

i. When the user chooses to, the program must be able to playback the data in the graphs.
ii. The data must, as with user mouse hovering, display a pointing device showing the values at that specific point.
iii. The speed the playback happens at must be selectable from a range of options.
iv. The user will be able to start the data playing back at any time point.
v. The user will be able to stop the data playback.

vi. The user will be able to take over graph hovering or zooming, and then let the data playback carry on its work.

4. User Interface;
   (a) A dark theme must be implemented across the system in order to reduce eye strain.
   (b) The user must be able to navigate the software intuitively.
   (c) There will be no new windows opening to show different graphs, everything must be kept on the same screen.

5. Technical;
   (a) The solution must be compatible with Windows, MacOS and Linux.
   (b) The solution must be ready to be upgraded;
      i. By using open source libraries.
      ii. By taking a modular approach to its seperate parts.

4.2 Methodology

4.2.1 CAN Network

As specified above there is no specification per se for the for the first part of the project, and therefore the methodology will largely be prototype based. The prototype methodology prepares systems which typically are not complete, with many of the details yet to build. The goal is to provide a system with the functionality intended so that it can be implemented later [47].

This presents itself with a large number of benefits, such as allowing the team members to get involved in the feedback process much quicker, allowing the development team to understand better to system rather than trying to "just make it work" as well as easier error detection and quicker debugging, both of technical and of the requirements.

Therefore, 'until further notice' this project would be tackled at a 'leisurely rate' - focusing on having a firm understanding of the CAN network concept by the end of Semester 2 Week 4. Meanwhile, the form of tracking of this project was reporting the progress made to the Head of Electronics so that he would be in the loop and would order parts if they were needed.

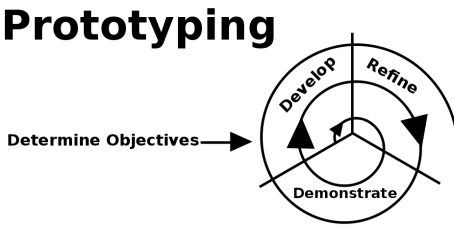


Figure 4.1: Prototyping Model [48]

4.2.2 Data Logger

The methodology for this project is exactly the same as the CAN network. A prototyping approach, perhaps with some more research aspects and with the need to communicate with the Electronics departments more. The timelines are similar, with the first prototyping stage also ending at around Semester 2 Week 4.

4.2.3 Live telemetry

So far there have been two important projects which have taken Prototyping approaches as they are also new parts for the team and the departments need to approach them with their own considerations. Therefore this project didn't take the time line, nor the imortance for other departments. This was also the the only project which, in one way or another, had to do with hardware where I was given completely free rein over the decisions I made on its development methodology.

I was given a small time basis to report back and therefore I chose a Sprint like approach where I would have x time to perform the implementation I wanted to. The time was plenty, however I had to make sure to stay in track. At the end the project felt like a mix between the fast paced atmosphere of a Sprint, but with the beginning-to-end approach of Waterfall.

I end up thinking of this project as a Minimum Viable Product, which is when a product has just enough features to provide feedback on future developments.

4.2.4 Telemetry Software

In this section we will look at different methodologies in order to assert the best one for this particular project.

4.2.4.1 Project Layout

The way this project will be laid out differs from the other methods in the sense that this deliverable will take no inputs from other departments. And therefore the focus will be on the communication with a client-like figure rather than with a more technically-inclined cooperative partner from another department.

This means that the development is completely up to me, as long as the results are shown off and feedback is allowed.

4.2.4.2 Choosing a Methodology

We will start by looking at the 'traditional' method to develop software, and then built up from there based on the client's requirements.

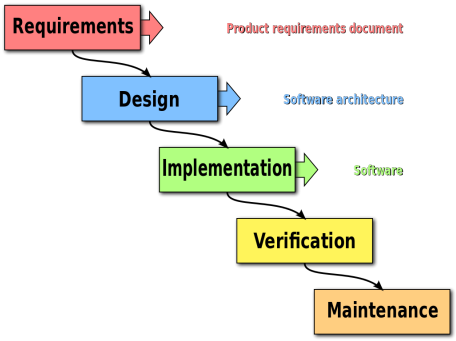


Figure 4.2: Waterfall Model [33]

The waterfall method is well known within the software development community, as it is easy to understand and ready for scalability - allowing anywhere from an individual to a team to use it. Furthermore it is great for projects which require great attention to detail and therefore need stage A to complete before moving to stage B.

All of those above facts paired with the method's simplicity for testing and deployment make it, at first, an appealing choice. Since it is easy simple enough that both the development team and the clients can understand it. However, it isn't a requirement (nor in some cases considered good practise) to allow the client complete transparency to the development method. The client should only give input when required by the developing team and in the case of waterfall, that is only at the beginning. [34]

Which brings us to one of the first flaws of this method, it does not allow for developers to go back to refine areas or process feedback given by the clients. This is one of LUMotorsport main

points for the development as they want to know what is going on with the software and may want to offer advice, feedback or simply see the progress. This point becomes stronger when we pair with one of the requirements, which is that the software must be built for upgradability. Waterfall will not do as we need something more flexible and that responds to small changes better.

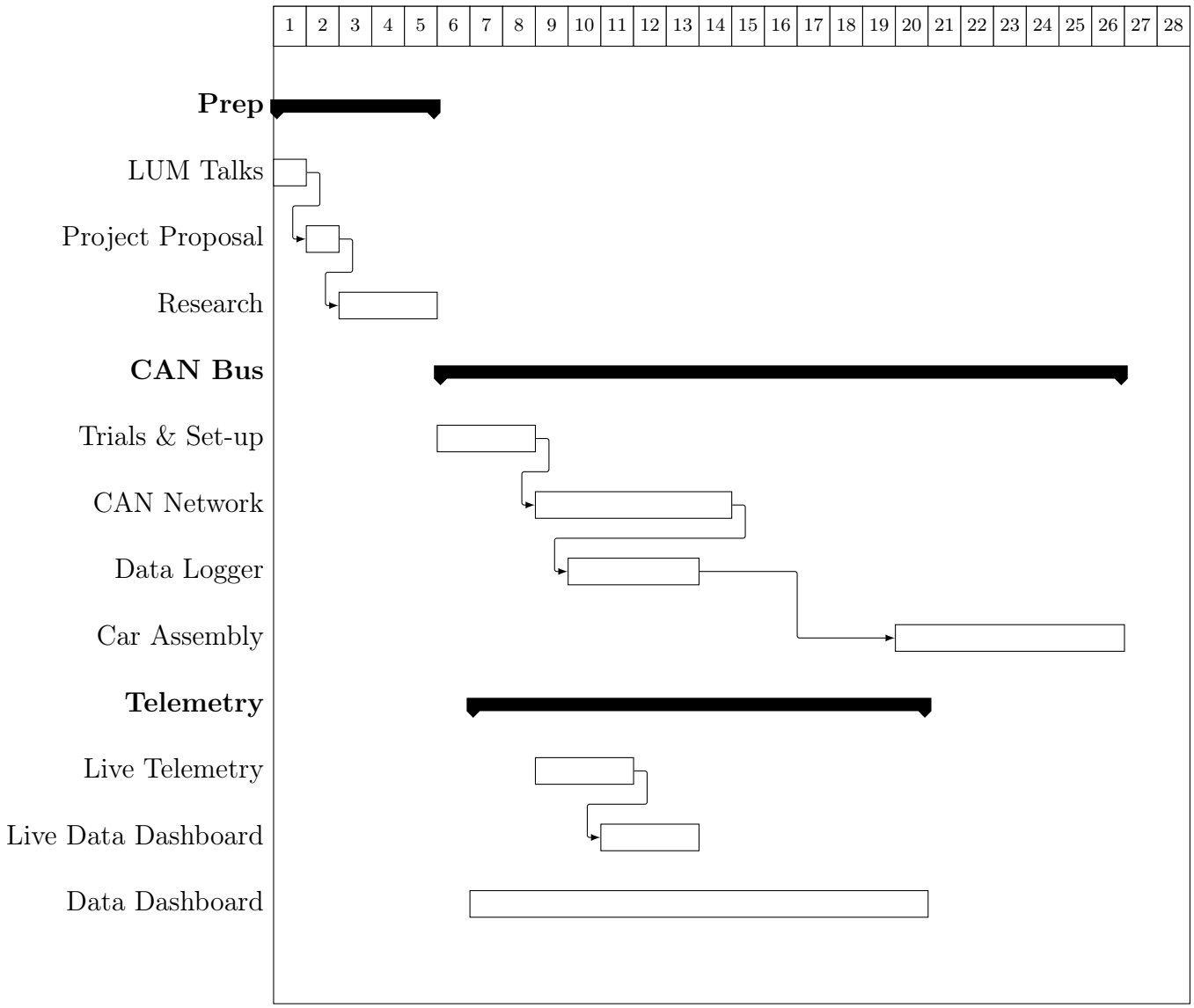
Therefore the process chosen for this particular project is a more iterative and quick evolving approach. A hybrid between the iterative approach and communicative manner of Agile method and the version-control philosophy of the more niche Incremental Build Model. This hybrid model will also change dynamically based upon problems encountered, taking a more proof-of-concept approach to any error which is encountered and cannot be solved after some minor tweaking or research - similar to the Prototype Methodology.

I am confident in this hybrid model due to the nature of LUMotorsport - a team which is geared towards a more technical approach and thus can help by providing feedback and understanding the software limitations. Furthermore, being part of LUMotorsport has made it 'harder' for me to think of them as the 'the client'; this is also helped by the hybrid model as it does not necessarily stick the formality of Agile and can sometimes take the quick development non-communicative approach of the Iterative model when no input is needed.

4.3 Time Management

Here is the original Gantt chart which was planned to be followed. However, after the COVID-19 lockdown there were changes made to the structure of the projects.

This chart should be taken as a notice of intention rather than an actual layout of the progress due to the rapidly changing environment of this last months.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
Prep
LUM Talks
Project Proposal
Research
CAN Bus
Trials & Set-up
CAN Network
Data Logger
Car Assembly
Telemetry
Live Telemetry
Live Data Dashboard
Data Dashboard

5. CAN Network

5.1 Design

5.1.1 Development Platform

For this project the platform being used was Arduino, a software and hardware open source company operating under the GNU License (which is "intended to guarantee your freedom to share and change all versions of a program" [49]). In terms of software, Arduino provides a microcrontroller board equipped with analog and digital I/O to allow for expansion of the board's features. The software area of Arduino is the Arduino IDE, which uses a heavily modified C++ version in order to import code to the board.

In essence, this means that using only an Arduino board a user has access to a computer that he or she can program, and once I/O is brought into the equation to capabilities of the Arduino development environment are enormous. Here is an example;

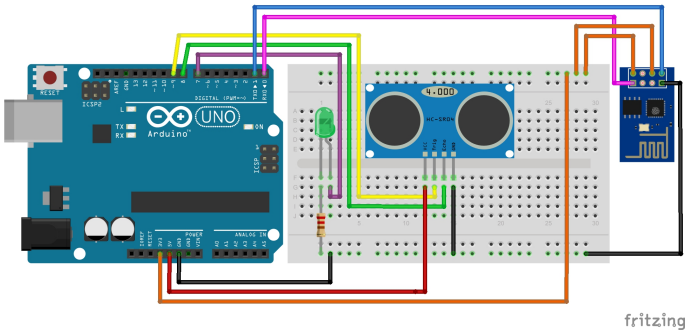


Figure 5.1: A Guinea Pig Surveillance System [50]

Figure 5.1 shows three components and a breadboard. The breadboard is used to temporarily connect modules together for prototyping - it is the white board. The big blue board is an Arduino that is connected to an input and an output - the input is the module with two black dots; an ultrasonic sensor and the smaller darker blue output is a WiFi module.

The ultrasonic sensor will send ultrasonic waves with one of the black dots (the emitter) and then expect the waves to bounce back and be registered by the other back dot (the receiver), this sensors are useful to detect motion. This sensor readings are passed to the Arduino over SPI and interpreted by the Arduino's CPU. When the Arduino decides that something is worth notifying the end user about it will pass information to the WiFi module, which will emit data to the end user's connected devices. This is just one of thousands of examples that Arduino has to offer, showcasing its modularity with such simple yet effective hardware.

5.1.2 Development Environment

The development environment for the first stage of the project would simply involve an array of Arduino Unos, which are the most popular Arduino board. Along with the actual CAN boards. We chose the MCP2515 Can Bus Modules, which are around $3 per piece.

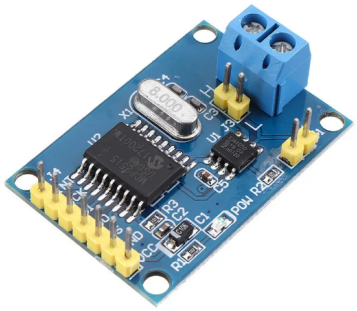


Figure 5.2: MCP 2515 CAN Bus Board

These boards have a CPU which processes the data into the CAN standard and therefore the 'only thing' the Arduino has to worry about is sending and receiving the data. This is incredibly helpful as there is no need to worry about programming anything that has to do with the CAN packet that isn't the data or the ID of the device.

5.2 Implementation

5.2.1 COVID's Influence

This is a message which may be repeated in other categories, both for this project or for other ones. Even though I have tried to only discuss COVID-19 above in Chapter 2.5.1, leaving the rest of this project without any COVID-19 messages - it is important to get clarify some development aspects.

Sadly due to the current situation the second stage of development could not be completed nor started. This means that only basic prototyping was able to be performed as part of this project - for which I apologize deeply, however I hope that it is understood. As part of Loughborough University's Engineering Department we must adhere to the regulations put in place, and one of those was the locking of the workshop where the car's development occurred. At this point in writing the workshop is still closed.

The workshop not only had the Arduinos and other hardware parts, but also the laptop used to develop most of the code, and therefore some code snippets may not be available as no one has been able to access the laboratory. The data is backed-up however not to anywhere in the cloud, just to an external HDD which also resides in the workshop.

5.2.2 Prototype 1 - Sending a CAN message

This was the equivalent of a "Hello World" message, a proof of concept iteration. I used, as explained above, the library specific for that CAN Board's CPU. The only line of code that required modification was the contents of the message being sent;

```
// Above this all I had to do was define the PIN layout and initialize the board
// and the Serial Baud Rate to enable Arduino-Computer USB communication
unsigned char stmp[8] = {0x0E, 0x00, 0xFF, 0x22, 0xE9, 0xFA, 0xDD, 0x51};
void loop()
{   Serial.println("In loop");
    CAN.sendMsgBuf(txID,1, 8, stmp);
    //(transmitting board ID, extended frame (true), message length, message)
    delay(25); //25ms
}
```

Here is the circuit layout, where the red and blue lines are the CAN High and CAN Low;

5.2.3 Prototype 2 - Customizing the CAN message

This version was about understanding the way the CAN packet treated the data, and how to start communicating more dynamically. I created a message that was 8 bits and sent it over as a combination of text and actual data that was collected from an input from the transmitting Arduino;

```
Vo = analogRead(ThermistorPin);
R2 = R1 * (1023.0 / (float)Vo - 1.0);
logR2 = log(R2);
T = (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2));
Tc = T - 273.15; // Reference A, check below
```

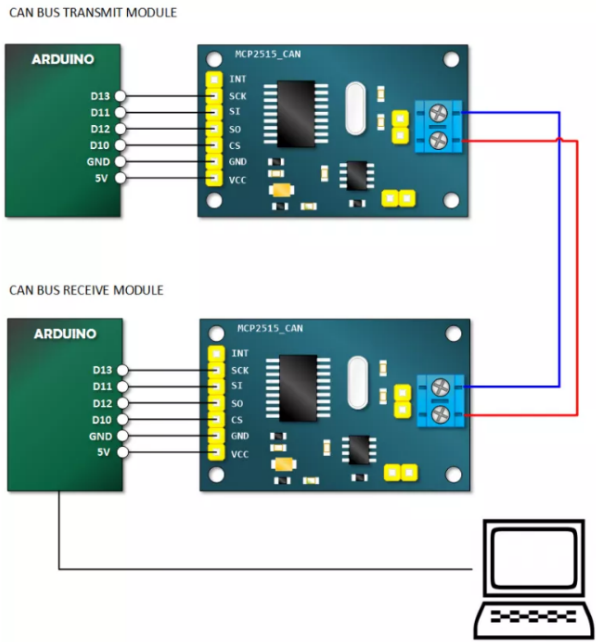


Figure 5.3: Simple Arduino-CAN-Arduino layout [51]

```
unsigned char stmp[8] = {ledHIGH, 1, 2, 3, Tc, 5, 6, 7};

CAN.sendMsgBuf(0x60,0, 8, stmp);
```

Reference A is [52]

5.2.4 Prototype 3 - Actuators over CAN

This version navigated the idea of creating an actuator for the receiving CAN board. The library had code that could run on the receiving board in the form of checking to see if a CAN message had arrived in the loop section. This was a continuation of Version 2 where the first index in the message buffer is `ledHIGH` only that in this scenario there was an LED connected to the receiving board so it could light up.

```
if(CAN_MSGAVAIL == CAN.checkReceive()) // check if data coming
{
    CAN.readMsgBuf(&len, buf); // read data,  len: data length, buf: data buf
    unsigned char canId = CAN.getCanId();
```

```
    Serial.println("-----------------------------");
    Serial.println("get data from ID: ");
    Serial.println(canId);
    for(int i = 0; i<len; i++)    // print the data
    {
            Serial.print(buf[i]);
            Serial.print("\t");
            if(ledON && i==0)
            {

                digitalWrite(LED, buf[i]); //changing the LED's state
                ledON = 0;
                delay(500);
            }
            else if((!(ledON)) && i==4)
            {}
                digitalWrite(LED, buf[i]);
                ledON = 1;
            }
        }
    }
```

5.2.5 Prototype 4 - Dual CAN Communication

One of CAN's main features is something that hasn't been yet explored, that is the ability to send and receive messages not only one or the other one. For this task two laptops were used, one connected to each CAN Arduino node in order to see their output and to aid in debugging.

This was quite straight forward; the only thing that had to be done was adding both CAN sending and CAN receiving code in the Loop functions. With the only consideration being the need to change the name of each CAN's ID. Find below a simplified code visualization of this;

```
// Sending
unsigned char stmp[8] = {random(120), temp, 2, 3, 4, 5, 6, 7};
    CAN.sendMsgBuf(0x60,0, 8, stmp); //change id per CAN node (0x60)
    delay(1000);

// Receiving
    if(CAN_MSGAVAIL == CAN.checkReceive())
```

{...

5.2.6 Next Prototypes

The following prototypes where part of the next two sections; "Data Logger" and "Live Telemetry" as they both used CAN communication and therefore will be covered more in those sections.

5.2.7 Rule Clarification

After some non-event weeks of Formula Student where the team (including myself) where 'getting on with our work' a big development happened. The rule clarification requested by the High Voltage team came back, it resolved a query the team had regarding the measurement of the battery cell's temperature and it stated that every other battery cell had to be measured for temperature changes with a rate no slower than 10Hz. The problem came with the realisation that half of the battery cells came to around 250 units.

This caused a team-wide meeting where the news were broken down and every department then had subsequent had meetings with the Technical Lead in order to establish the importance for this change. Regarding the CAN network the change was big, it was almost doubling the amount of sensors that the team expected to use for the car and therefore it was necessary to rethink the CAN network.

In a way this is what the prototyping was for, and my sentence some chapters above "This is because of the way the Formula Student development process happens and the added volatile nature of this year's powertrain change, from combustion to electric." became true. After much consideration there were two options considered;

- Creating more than one CAN network from Arduinos; this option might add some extra weight and volume to the final build, however it is a common practise in the automotive industry to use separate CAN networks, and to separate them based on their critical level (Safety, Entretrainment, etc..etc..)
- Use a more powerful solution than an Arduino, such as a Rasperry Pi; which is an Linux based computer used for projects similar to Arduino but with a lot more power and slightly harder to understand functioning (as it includes an operating system and by definition it can perform more tasks)

By the time these two decisions where presented to the Electronics team, the COVID pandemic forced the workshop to be closed down and the Formula Student event for the 2020 Summer was cancelled, therefore completely shifting the priorities of the car's development.

Sadly no more work was done on this.

5.3 Results

The results of integrating a prototype version of the CAN network proved to be a fun project, which was slowly gaining traction as more and more modules where being added to the Arduino nodes. These parts communicated over SPI until they reached the Arduino, and once at the Arduino they were sent across the car using CAN. This combination was easy to understand, taking advantage of the both communication protocols.

In the next chapters the full work done with CAN in regards to Data Logger and Live Telemetry Integration will be outlined, however it is fair to say that those areas also proved to be of a modular approach, which eased development.

6. Data Logger

6.1 Design

6.1.1 Development Platform & Environment

This project uses the Arduino platform, as well as the developments made in the CAN network. As a new incorporation to the modules used as a part of Arduino, there is the SD card reader. This $5 module gives Arduino's output the ability to be recorded in a storage location.

Figure 6.1: SD Card Reader [53]

The environment is the same as the one that has been established for the CAN project, and in fact at some points in the day CAN network and Data Logger development will happen at the same time.

6.2 Implementation

6.2.1 COVID's Influence

This is a message which may be repeated in other categories, both for this project or for other ones. Even though I have tried to only discuss COVID-19 above in Chapter 2.5.1, leaving the rest of this project without any COVID-19 messages - it is important to get clarify some development aspects.

Sadly due to the current situation the second stage of development could not be completed nor started. This means that only basic prototyping was able to be performed as part of this project - for which I apologize deeply, however I hope that it is understood. As part of

Loughborough University's Engineering Department we must adhere to the regulations put in place, and one of those was the locking of the workshop where the car's development occurred. At this point in writing the workshop is still closed.

The workshop not only had the Arduinos and other hardware parts, but also the laptop used to develop most of the code, and therefore some code snippets may not be available as no one has been able to access the laboratory. The data is backed-up however not to anywhere in the cloud, just to an external HDD which also resides in the workshop.

6.2.2 Request 1 - Digital Channels

This section was partly covered in the CAN project as some digital data was passed to the CAN network.

It is worth mentioning that it is currently unknown the amount of digital channels that the Arduino that is hosting the SD card will receive. This can be a problem, as Arduinos have a limited amount of I/O, with boards like the Arduino Mega presenting more I/O but still not fixing the problem. A solution of this can be buying what is known as a Shield board. These stack up on top of the Arduino and add functionality to it, a common Shield module that is available is one that increases the I/O.

However this can still be limiting, as the standard shield will only add from 15 to 30 extra analog or digital inputs. We can further increase capacity by shifting away from Arduino specific components and pick a generic Integrated Circuit (IC) - like the MCP23S17, a 16-Bit SPI I/O Expander. This will give the Arduino 16 more inputs and the ability to connect 8 IC's together, for a total of 108 inputs.

From this point and in order to increase capacity more whilst still keeping the setup 'clean' there might be a need to switch to a secondary Arduino, which could log other data. This would resemble the 2 or more CAN Network layout that was described in the CAN Network section. Apart from that, a Rasperry Pi might be considered. However this was never formally looked at.

6.2.3 Request 2 - ADC

After talking with the Electronics department on one of the meetings it was decided that the ADC would be something that they would built and that therefore all the input would be passed to be as Digital.

6.2.4 Request 3 - Data Treating

As we are using an Arduino as the SD Card Reader's input, it becomes much more straight forward to input custom parameters or to treat the data in a certain way. For example; if I wanted to implement a custom parameter in front of a certain data entry at would be as easy as;

```
File dataFile = SD.open(FileName, FILE_WRITE);
  if (dataFile) {
    dataFile.print(buf[1]);
    dataFile.print(",");
    dataFile.print(buf[4]);
    dataFile.print(", parameter: ");
    dataFile.println(buf[5]);
```

6.2.5 Request 4 - Loading Memory

On an abstract level this seems like a complicated matter; but it is handled easily by the Arduino hardware. An Arduino has three memory types [54];

- Flash memory (program space), is where the Arduino sketch is stored.
- SRAM (static random access memory) is where the sketch creates and manipulates variables when it runs.
- EEPROM is memory space that programmers can use to store long-term information.

In this case it will be the EEPROM (Electrically Erasable Programmable Read-Only Memory) that holds the operating instructions (loaded from the Arduino IDE) until it is erased by new instructions (for example a new code iteration).

6.2.6 Request 5 - Offloading Storage

Once again, a simple question which was answered as part of another area - the storage used to hold the data will be an SD card. SD cards are widely used for storing short term data in embedded systems that require the user to retrieve them, for example dashboard cameras. The only 'real' alternative would be something like a Solid State Drive as it also has no moving parts.

6.2.7 Request 6 - Power supply

After talking with the Electronics team I was informed that the Arduino would be powered of the car's battery for Low Voltage which would give power to other low voltage items such as sensors and other integrated circuits.

6.2.8 Data Logger CAN Code

Unfortunately this part of the code is one of the pieces of that I wasn't able to get my hands on, since the only copy is in the workshop. However a pseudocode version might look like this;

```
sendingArduino{
    function loop{
        sensorValue = readPinX
        CAN.send(sensorValue)
    }
}

receivingArduino{
    function loop{
        for line in lines{
            sdCard.save(line)
        }
    }
}
```

6.3 Results

The results for the first of the development where hopeful, however once again the COVID-19 lockdown meant that no more work could be done. Below is a table with the exploratory requirements set up as the first stage of the development. It is a more informal approach to a testing and results table, however that is because these specification was never meant to be treated as a formal document - rather an allowance of time for me to get comfortable with Arduino, the CAN network and the data logger. A green box denotes a satisfactory result to the coding/researching, whilst a red shows that the task wasn't completed and yellow shows a completed task but with limitations.

1. Treating digital channels

   Found a way to take 100+ inputs, however no real solution for a larger number was found. Of course, what ever the solution is, will be hard to find as I am still pending on the system's description in regards to sensor capacity from the Electronics and High Voltage teams. Nonetheless, as a prototype or proof-of-concept there would be nothing wrong with having several 100+ CAN networks logging in different Data Loggers.

2. Treating analog channels and ADC creation

   After communicating with the Electronic Team regarding analog signals I was reassured that the ADC (Analog to Digital Converter) process would be handled on their side.

3. Micro-processor to treat incoming data

   Arduino gives us the chance to use its processor to handle the data before it goes into the SD card.

4. Memory to load

   Arduino's EEPROM and Arduino' IDE give us the chance to save data on a non-volatile state, and to only delete it when new instructions are being uploaded.

5. Storage to offload data

   The only 'real' option found; in terms of cost, size and Arduino compatibility was an SD card.

6. Power supply

   After talking to the Electronics team it was confirmed that the power supply would be a battery on the car, therefore the SD card reader and its Arduino don't need any extra source of power.

7. Live Telemetry

7.1 Design

7.1.1 Development Platform & Environment

After some research at the different methods of implementing real-time telemetry at a low cost I ended up narrowing it down to two possibilities; Radio or WiFi.

Radio was a piece of hardware we had laying around, it was bulkier and harder to understand - however it is the go-to when it comes to remote communication in areas like Arduino. It also allows for the configuration of aspects which leads to changes in the range, data throughput and reliability. However it was quite hard to get started, with the feedback for the hardware almost negligible.

Furthermore, there have been talks that some Formula Students may strengthen regulations on radio frequencies, that added with the fact that every Formula Student event states in its rules that it cannot be held liable for radio frequencies not working due to local regulations makes it a hard choice, at least as the only system in use - perhaps if it was coupled with WiFi it would be different. But then coupling two different systems together and treating one as a fallback of the other one is hard to maintain and to justify to judges.

On the other the other hand there has been a fairly recent surge in demand, production, libraries and community support for the ESP8266 - a $6 WiFi microchip with a full TCP/IP stack which allows its uses to be extended to a variety of helpful functions (especially in home automation) such as a server or WiFi repeater.

The ESP8266 also comes with its own microchip which connects to the computer via micro-usb and can use the Arduino IDE along with a large amount of libraries - this would make it an interesting option as a radio module would need an extra Arduino. Not only would it need an Arduino, but also it's own - as we discovered that our radio module wasn't co-operating with the other SPI devices attached.

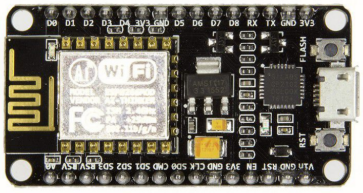


Figure 7.1: ESP8266

7.2 Implementation

7.2.1 COVID's Influence

Sadly all of the code but a small part for this section has been left in the workshop, therefore the Implementation will be shorter and only explain the key points.

7.2.2 Radio

Two Arduinos were plugged in to individual computers, each one with their own Radio antenna. A message was meant to ping back and forth the radios but without any success. Slowly, the problem was debugged; as the Arduino was stripped of the other modules it had and the radio's code was set for one to receive and one to ping.

After those adjustments it worked, however which much difficulty. There were suspicions that the radio module might be broken within the Electronics team, who kindly donated the radio module previously.

7.2.3 WiFi and custom server

After that, the ESP8266 was used. Right away it was more straightforward to use, with better examples and online references and actual verbose output if any problem was encountered. A WiFi hotspot was made called "LUM WiFi" and a website was hosted on `192.168.1.4`.

After that, a random value was passed with a quickly made HTML auto refresh feature every 1 second, that worked too. It seemed as if the progress was good. The next feature was to try and implement a graph, however I bumped into some problems there as the ESP8266 did not have enough memory to store the JS of even the smallest graphing library.

7.2.4 Radio and MATLAB

What the ESP8266 could do was pass a .JSON file of data that refreshed constantly and then allow any connected computer to refresh that data and plot it as a graph. As a quick test, a MATLAB script was made which would grab the data and then output it as an updating chart; a built-in MATLAB function. Here is a short part of the script;

```
while true
    WebsiteData = strsplit(webread(url),['"', "'"]);
    figure(1)
    addpoints(h,i,str2double(WebsiteData{3}));
    addpoints(TempLim,i,MaxTemp);
    drawnow
```

This seemed to work so well in fact that the Technical Lead was happy to leave it like that, it had the advantage that since it was MATLAB the team could interact with the code better, since most LUMotorsport team members have experience in MATLAB.

7.2.5 Testing Range

Unfortunately, when it was time to test the range the lockdown happened and I was never able to get some results for that

7.3 Results

I am confident that, almost by a stroke of luck, the MATLAB solution could be one that the team could built upon. I would have liked to buy an antenna for the ESP8266, which are also cheap, in order to try and extend the range as I imagine the built-in antenna will not work too well over long distances.

8. Telemetry Software

8.1 Design

8.1.1 Development Platform

Choosing the development platform turned out to be a process which, whilst insightful, was time consuming. There were 3 different options which where considered after the research made.

8.1.1.1 MATLAB's App Designer

MATLAB is a programming platform designed specifically for engineers and scientists. It is the language the most LUMotorsport team members have a grasp, as they are all engineers. They will use the language specifically to create visual representation of complex data readings from the car, those that MoTeC's telemetry software might struggle with. MATLAB also allows the team to manage large engineering systems and their logic by providing them a rich and customizable GUI.

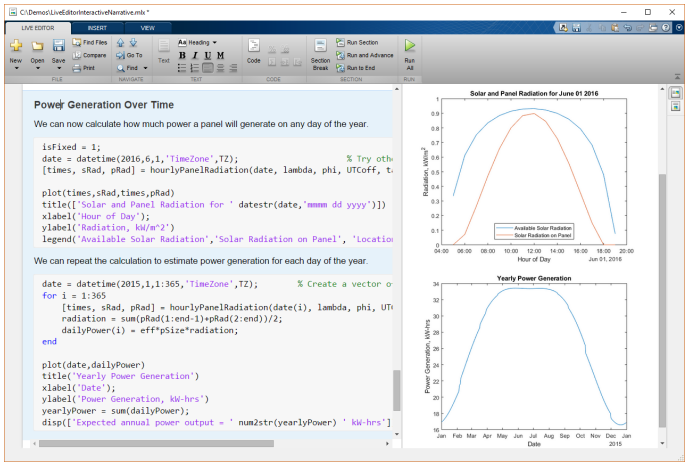


Figure 8.1: The MATLAB Environment [35]

Apart from what was mentioned above MATLAB also offers the App Designer. A toolkit to develop a user interface with custom input methods (sliders, buttons, text entries) and graphing capabilities. After that the software can be 'published' and it would appear on MATLAB's 'Apps' Tab whenever the software was used.

Using MATLAB was the first idea and it presented itself with a large number of positive aspects;

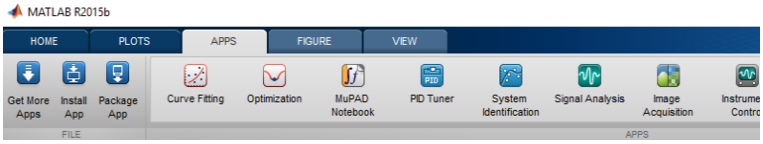


Figure 8.2: Accessing the custom app [35]

- Everyone in the team understands MATLAB and therefore the team would be able to greatly interact with the software and quickly draw custom charts.
- Since MATLAB's purpose is so 'niche', quick starting something like a graph is extremely quickly.
- The App Designer was made with the idea of handling datasets and therefore no additional libraries would be required.
- The software can run based on the team's MATLAB subscription and therefore easy to deploy to multiple computers.

Upon further investigation however, several critical faults where found with the software;

- In terms of the software's philosophy, its scalability seems to be a limiting factor - it seems that all programs displayed in demos, examples and tutorials seem to do one or two functions, and do them well. This does not align with LUMotorsport's requirements list.
- Furthering on from the point above, it seems as if the software is still 'young' and not refined - having lots of bugs, with one user even acknowledging that *"over 800 lines of code is known to cause a slowdown"* [36] to the point of it being unusable.
- The App Designer itself is clearly thought out to be quick scripting program, rather than something ready for a larger project. Another example of this is that when going from the 'Layout' view to the 'Code' view - large chunks of the code cannot be edited, as if the user needed pointing to the areas that he can edit, which are only graph specific properties. It is something that I struggled getting along with.
- From a user's end of point, I believe that MATLAB's quick charting abilities might, in the long term, cause a slow down to productivity. If everyone knows the MATLAB environment, then everyone can change the code 'here and there' in order to suit it for a purpose and then never change it back - this will end up in a folder with endless "SOFTWARE 1 COPY FINAL copy 2"-type files. This thought is reinforced by my experiences within LUMotorsport.

- MATLAB is a closed-source program and the team gets it for free thanks to the "university license". Nonetheless, with the abundance of open-source software and their quickly developing libraries, it seems like using something that (at least for now) won't receive user feedback could be a waste.
- The last issue was centered around MATLAB's documentation and examples structure, which I found to be lacking and not nearly as insightful as a software of that caliber should be able to provide.

The overarching believe of reviewing App Designer was that it was some years away from having the user following, pedigree and usability as other software. Despite that, it was hard to dismiss as an alternative due to the team's closeness to it.

After a meeting with the Technical Lead, we arranged for the fact that no matter what we chose, it would always be a hybrid - going hand in hand with MATLAB when in need of making a smaller, more complex graphing script. This tends to happen when the team is making electrical calculations or when individuals from the team are using the car for one of their final year projects.

8.1.1.2 R, RStudio and RShiny

The R programming language is similar to MATLAB in the sense that it is used by computer scientists and engineers above all, however R focus shifts away from algorithm and model development and instead focuses more on statistics. This isn't a problem as both R and MATLAB are extremely good at representing graphical solutions.

Apart from R a user can get RStudio which is R's Integrated Development Environment (IDE). IT is widely used in industry, with names like Walmart, NASA, General Electric, eBay, Standander and HONDA in it's community. A lot of the success R is having is in fact coming from its community, since the whole R project is open source there are plenty of libraries and examples to view. R and Rstudio believe that *"free and open source data analysis software is a foundation for innovative and important work in science, education, and industry"* [37].

One library that is widely used and accepted as one of R's best areas is the RShiny library, which is part of RStudio. It allows users to interact with the data models rather than just viewing them. This is achieved by RShiny's reactive programming model - similar to that of a spreadsheet software [38]. RShiny's reactive tools and graphs are built as widgets, which make the software even more modular and to top everything off, the software can the be built as a stand-alone web app and even be deployed from a custom server.

Furthermore, one of Shiny's extensions (again, thanks to R's open source approach) is RShinyDashboard - which allows the widgets to be ported to a dashboard-like software, which easily allows the user to start interacting with things like tabs, pages and multiple graphs.

As stated above, there are a lot of good things that would come from R - in short;

- To start with, RStudio, at least compared to MATLAB and similar software, is the reason why the computer science field is engaging and fast growing. It is, from my experience, a love letter to open source software and to its community. Developing in an environment like that makes those in it feel safety, knowing that the language's future is safe. Compared to MATLAB; who recently rebranded another app to become the now known App Designer.
- Although MATLAB allows for web apps too, they seem slower and harder to set up working - RShiny is built specifically for web apps.
- MATLAB's software is the accumulation of 15 years of work and upgrades, back when the field was very different. In terms of performance, MATLAB is quick to render, however when dealing with interactions the R project seemed to grain a strong advantage against R.
- The availability to load the RShinyDashboard studio to a server seems like an interesting future proposition.
- The documentation, examples and community engagement from R seems to be right where one would expect.
- In terms of what the end product would be, it seems as if MATLAB would be a collection of small scripts that the team has to open either from a subsection of the MATLAB software itself, or as 'clunky' web apps. RShinyDashboard would make the software feel like what it is meant to feel like, a finished, solid, intuitive, single program with a responsive user interface.

Again, like it was with MATLAB it seems as if there are all goods and no bads with the R project. However, with MATLAB there were a plethora of 'small' issues which piled up. What was interesting about R is the fact that there were two big issues;

- R is not covered in the basic Computer Science course, nor the Computer Science and Management (which is the one I am enrolled in) - t isn't covered in Engineering subjects either. This means that unless someone has had a personal/placement project with it before, the R project is meant to be first learned and then implemented. This causes

a dilemma; I will have less time for the actual development of the telemetry software if I choose to use R, apart from that I will need to document the code so that not only it shows the user how to change it but also how to understand it. This then will put the team in a spot where, in order to upgrade some areas (a few years down the line) someone will have to take time out of other things in order to re-write large sections of the code.

- This last point is similar to the one above - but less theoretical and qualitative and more quantitative based on my experience. The R platform and its extensions are hard to learn. They are hard to get started with, much harder than any MATLAB code. In its defence; the time spent VS work output curve gently shifts back from MATLAB back to R as the code starts to 'pile up' - this is what R was designed for.

8.1.1.3 Electron-based App

Finally, we look at Electron - a cross-platform environment which allows HTML, JS and CSS to be made into native-like applications based on Chronium and NodeJS. After further inspection it becomes clear that this might be a good compromise between the 'futuristic' R and the 'traditional' MATLAB;

- Electron is also open source, with an even larger community and user base than RShiny.

  Of course this popularity is thanks to the already existing web development user base, but perhaps that is also a positive point. As it means that there will be more community resources out there. Electron is currently being used for applications like Visual Studio Code, WhatsApp, Slack, Discord and Skype.

- An Electron program is, for the parts that the team want to change, an HTML and JS document. Therefore improvements to it could be a implemented a lot quicker and without the need for learn a completely new language.

- Apart from this key difference with the R project, Electron gives us the same main benefits.

And for the large list of advantages it gives us, there aren't too many disadvantages [40];

- Resource usage is a topic that keeps popping up in the community, as every Electron app runs its own version of Chromium (which is around 20 million lines of code). Electron apps can be considered to use too many resources, with a special focus on RAM. However

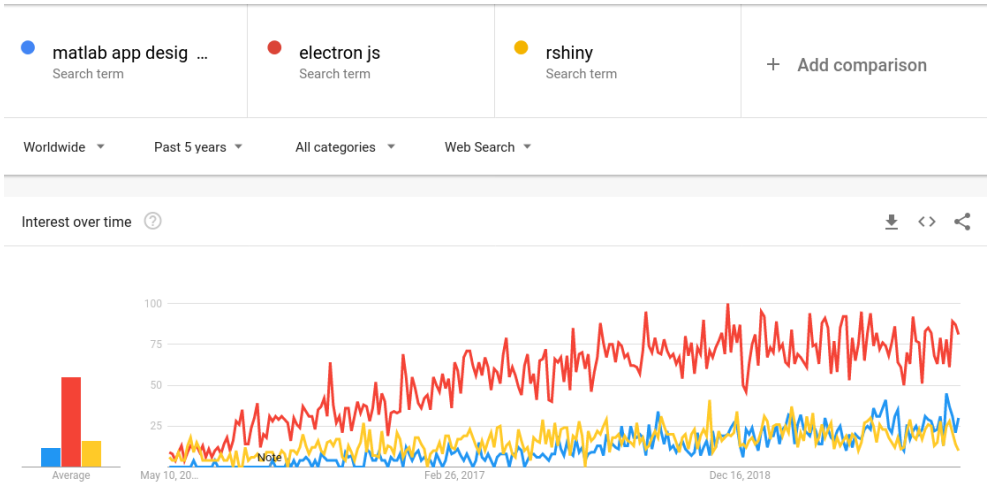


Figure 8.3: Popularity per Google Search [39]

this theory seems to be very dependant on the system being used and, above all, on the function of the app. Thankfully, the computers the university provides LUMotorsport with are powerful machines ready to run Computer Aided Design (CAD) software, so this worry is not a big one.

- The second complaint comes from a security aspect, both for the user and for the code itself. However we will not go to far into this topic as the telemetry software does not use logins or any critical information and it is intended to be tinkered with by the team.

8.1.2 Development Environment

For the development of this application I will be using an 8th generation Intel Core i5 CPU with integrated graphics and 8GB of RAM (ThinkPad X390) running Arch Linux. This is my daily driver computer and operating system, and therefore I feel comfortable developing in it.

For the code editor I will use Atom, which is also written in Electron. It has a dark theme, auto-completion, tab stacking, vim-like functionality, a file explorer and is upgradable with thousands of community made packages. This set up is paired with a recently acquired 31" 16:10 2560x1440 IPS screen. Which will allow me to tackle a large amount of tabs in order to work with different areas of the code.

For version control I will use a recent small project written in Bash, which I hope to upgrade as the project goes by. It is simply a tool which replicates the code into a new directory, allowing for the next version to be developed - however it also saves some self-made information for me

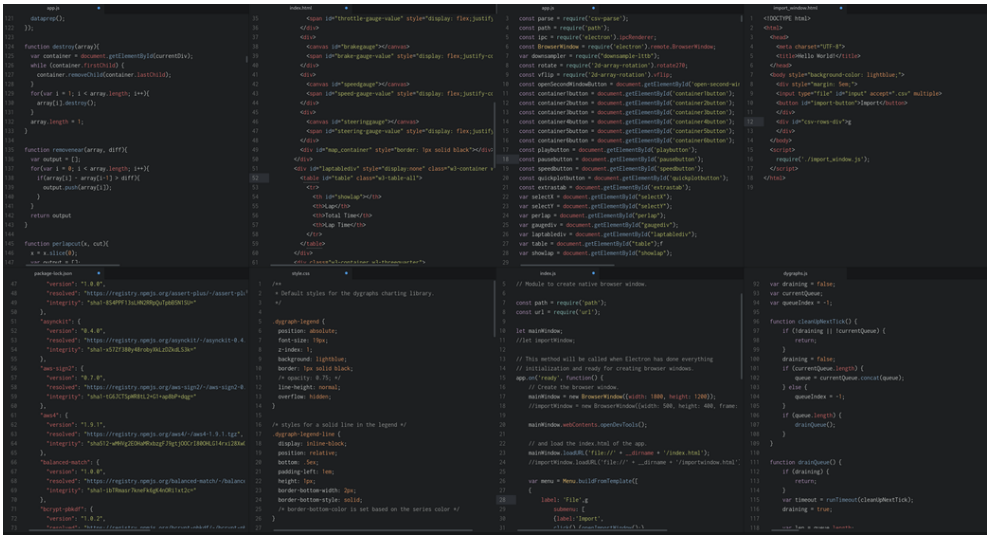


Figure 8.4: My 'debugging' Atom setup

to go back and look at the code, in order to ease the process of documentation both for this report and when showing LUMotorsport the progress made.

8.1.3 Prototyping layout

As I have been part of LUMotorsport for some years, having interacted with the telemetry software from the 'track side engineer' view and from the 'driver' view, I was fairly confident in what the layout of the software should be.

Therefore this stage was quick and when presented in a meeting it was authorised as the 'design to go for' without any additional considerations.

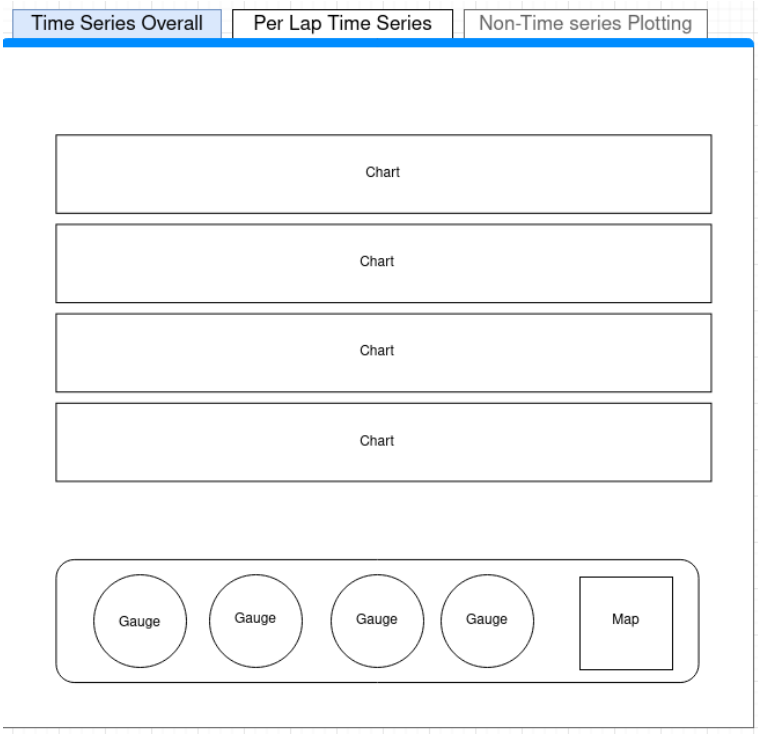


Figure 8.5: Time-series Full Run

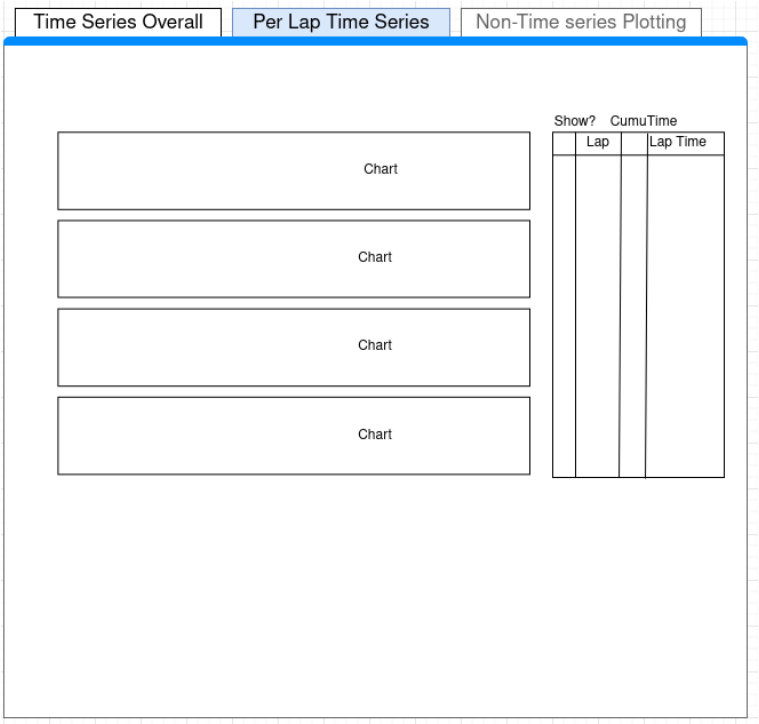


Figure 8.6: Time-series Per Lap Runs

Figure 8.7: Non time-series Plots

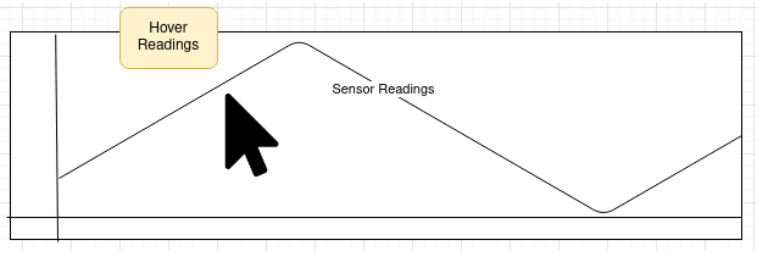


Figure 8.8: The mouse's interaction with the graph

8.2 Implementation

This section will outline the version by version increments from the development process. Sometimes there will be version jumps, however this will be to avoid any version which didn't see a lot of progress. Each section hopes to provide a brief description into what each different part contributed to the final deliverable.

8.2.1 Version 1 - Hello world

This section showed simply that I had correctly set up Electron. When opened it showed "Hello World! You are using Node 12, Chromium 80 and Electron 8" as part of the HTML code.

One of the good things Electron has is an easy to debug verbose system, this is thanks to the fact that it works like any other website would - in this version I also opened Dev Tools.

```
function createWindow () {
  // Create the browser window.
  const mainWindow = new BrowserWindow({
    width: 800,
    height: 600
  })

  // and load the index.html of the app.
  mainWindow.loadFile('index.html')

  // Open the DevTools.
  mainWindow.webContents.openDevTools()
}
```

8.2.2 Version 2 - Experimenting with graphing libraries

The first thing to do was find the right graphing library. After some research I found that the two main ways a web browser will render a graph, or any graphic for that matter (unless WebGL-level complex), will be either as .SVG (Scalable vector graphics) or a Canvas element. However, because SVG runs using the Document object model (DOM) it will become slow [41]. This is one of the reasons I wanted to make sure that whatever I chose was rendered as a Canvas, not SVG.

I found CanvasJS, a canvas-based interactive graphing library, and my first graph was made with this library.

```
var chart = new CanvasJS.Chart("chartContainer", {
        theme: "light1",
        animationEnabled: true,
```

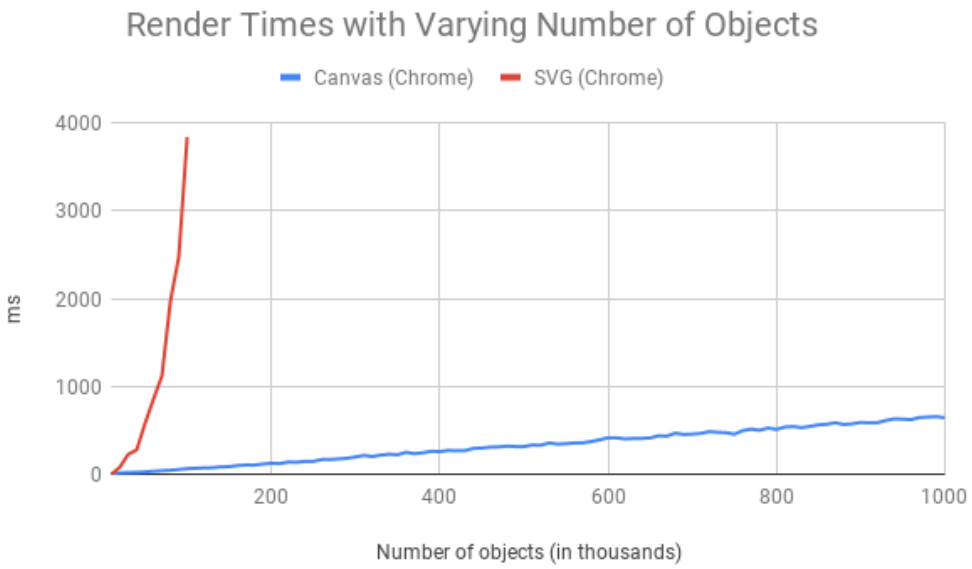


Figure 8.9: SVG vs Canvas [42]

```
title:{
        text: "Basic Column Chart"
},
data: //here was an array of data
```

8.2.3 Version 4 - Changing libraries

CanvasJS had been, from the beginning, a payware library. However it offered free licenses to university students and I believed that an exception could be made whereby a license would be attached a club (LUMotorsport) rather than a student (myself) in order to ensure that the software after I was gone.

After CanvasJS got back to me it was clear that this would be hard to arrange and therefore I continued my search for a graphing library with my new adquired knowledge from how straightforward CanvasJS was for plotting a graph.

I stumbled across a library called Chart.js and one called Dygraphs. Chart.js was very pleasing to look at and much more known than Dygraphs, however after some more research it was found that Chart.js struggled with large data sets, whilst Dygraphs was meant to be a performance above all library, that still had interactivity options like CanvasJS.

From now on, I will be using Dygraphs.

8.2.4 Version 5 - Dygraphs Test

A test of Dygraphs was performed and it was just as simple or even simpler to use than CanvasJS.

The library counted with only one tip of plot, however with a lot of theming options. It was also possible to interact with the chart by click dragging the cursor an area would become zoomed in.

8.2.5 Version 6 - Dygraph's Killer Feature

Dygraphs had built-in chart synchronization, which is when two or more graphs can become 'one' by sharing the same curson interactions.

```
g1 = new Dygraph(
  document.getElementById("div1"),
  "dataA.csv", // path to CSV file
  {}           // options
);
g2 = new Dygraph(
  document.getElementById("div2"),
  "dataA.csv", // path to CSV file
  {}           // options

var sync = Dygraph.synchronize(g1, g2); //an array can also be passed
```

8.2.6 Version 7 - Adding .CSV reading

In this version Dygraphs was able to read from a CSV file thanks to an NPM library called "csv-parse"

```
var parser = parse({delimiter: ','}, function (err, data) {
  var one_raw_1d = [];
```

```
  var one_raw_2d = [];
  var two_raw_1d = [];
  var two_raw_2d = [];
  var graph_ready = [];
  data.forEach(function(line) {
    var fulldata = { "time" : line[0]
                   , "wheels" : line[1]
                   , "throttle" : line[2]
                   , "brake" : line[3]
                   , "rpm" : line[4]
                   , "steering" : line[5]
                   , "glat" : line[6]
                   };
fs.createReadStream(inputFile).pipe(parser);
```

8.2.7 Version 10 - Dynamic Graph creation

In the past I was having to create the elements for the graphs to go into, however after this version I was able to add elements dynamically depending on the size of the graph.

```
for (i = 0; i < data[0].length; i++) {
   if (i < data[0].length){
     graph_ready.push(raw_1d[i].map(raw_2d => raw_2d.map(Number)).slice(1));
     //this takes away the first row of data, which are the headers
   }
 }
for (i = 0; i < graph_ready.length; i++) {
    document.getElementById('container').innerHTML+=('<div id="graph' + i + '"></div>');
}
var creations = [];
for (i = 0; i < graph_ready.length; i++) {
    creations[i] = new Dygraph(document.getElementById("graph" + i),
                   graph_ready[i],
                   {legend: "always", //this are the options
                   animatedZooms: true,
                   labels: [labels[0], labels[i]]
                   });

Dygraph.synchronize(creations, { //creations array being passed
```

```
  zoom: true,
  selection: true,
  range: false
});
```

8.2.8 Version 11 - .CSV from a file

An input was created so that the .CSV file read could come from a file rather than a static file in the code. This is how the team will interact with the code

```
    <input type="file" id="input" accept=".csv" multiple>
```

```
const inputElement = document.getElementById("input");
inputElement.addEventListener("change", handleFiles, false);
function handleFiles() {
  const fileList = this.files;
  inputFile = fileList[0].path;
  graphing();
}
```

This is how the software looked at to this point

dataB.csv is a dataset which includes a small self-generated set of values in order to replicate the car's readings.

8.2.9 Version 14 - Allowing custom tabulation

When making the charts the user is now allow to choose which tabs the data goes to, this is a first-time attempt at data organization. Which is one of the points in the specification which hadn't been dealt with yet.

```
var layoutOrder = [];
for (i = 1; i < graph_ready.length; i++) {
    if(tabSelector[i] == "tab1"){
      document.getElementById('container_tab1').innerHTML+=('<div id="graph' + i + '"></d
      layoutOrder.push("tab1:" + i);
```

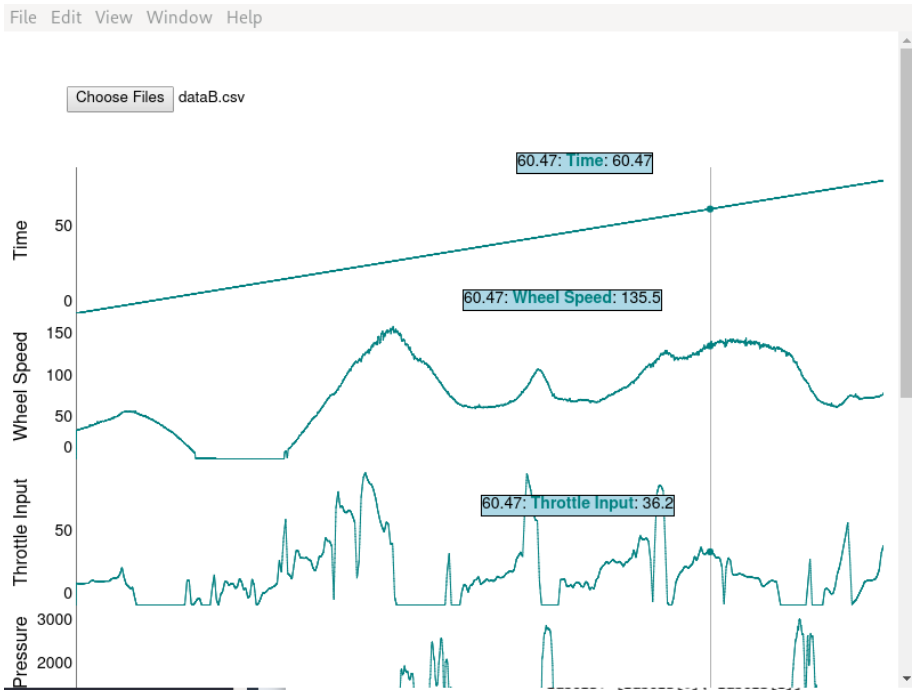


Figure 8.10: Version 11

```
    }
    else if(tabSelector[i] == "tab2"){
      document.getElementById('container_tab2').innerHTML+=('<div id="graph' + i + '"></d
      layoutOrder.push("tab2:" + i);
    }
}
```

Another window also opened when reading the data from the menu bar, so that in the "Import" button wasn't always visible from the main software.

The menu bar is one of Electron's functionality pieces.

```
var menu = Menu.buildFromTemplate([
  {
      label: 'File',
          submenu: [
          {label:'Import',
          click() {openImportWindow();}, //importwindow.js
          accelerator: 'CmdOrCtrl+I'},
```

```
            {type: 'separator'},
            {label:'Exit',
            click() {app.quit()},
            accelerator: 'CmdOrCtrl+Q'}
            ]
        }
    ])
    Menu.setApplicationMenu(menu);
```

The file 'importwindow.js' then took over as the file that read the .csv file.

For communication to happen I used Electron's ipcMain, an Event Emitter which handles asynchronous and synchronous messages sent.

```
//on importing window's JS
ipc.on('message', (event, message) => console.log(message));

//on mainWindow's JS
const ipc = require('electron').ipcRenderer;

//on electron's config JS
ipc.on('reply', (event, message) => {
        console.log(event, message);
        mainWindow.webContents.send('messageFromMain', `\${message}`);
})
```

8.2.10 Version 16 - Unexpected performance issues

At this point in time, the program, whilst not finished, was looking good. It had the ability to show synchronized data being read from a .CSV file and it could also position data in different user-selectable tabs. At this point in time I request from LUMotorsport some real data in order to see how the software would handle more data points.

A car logged file is 108 channels (sensors) being recorded at a rate of 100Hz. This means that every 0.01 seconds of data the software has to plot 108 channels.

I was sent 200 seconds of data from an engine start some months ago, and much to my surprise the graph took around 40 seconds to render. This kind of performance was impossible

to deal with, and so began the largest debugging session this piece of code would encounter for its development life.

8.2.11 Version 17 - Cleaning out the data retrieving

After looking back at the code and trying out a variety of elements, I finally stumbled across a big part of the reason of the slow down. It was the way I was treating the data from a .CSV format; I was doing several object to array and array to object conversations that were not necessary. They were simply there as part of the code getting bigger and no considerations for its performance being put in place.

Here is an example of the kind of inefficiencies that were eradicated.

```
  for (i = 0; i < Object.values(fulldata).length; i++) {
      if (i < (Object.values(fulldata).length)){
        raw_1d[i].push([Object.values(fulldata)[0], Object.values(fulldata)[i]]);
      }
    }
});
```

Functions like the one above that convert Objects and push them to arrays where removed, and instead replaced by much cleaner 'only array' code. This removed the slowdown (with the 200 second piece) by about 50% which is an incredible improvement.

However the issue remained, as the data got larger (and it would, up to 2400 seconds or 40 minutes) the inefficiencies would grow again. After the code was cleaned up as much as possible in terms of data preparation it was the graphing library the one that was slowly down the program.

8.2.12 Version 21 - Downsampling the data

After many different versions trying different things; for example only showing the data in segments of x seconds and letting the user pan across the different sections with buttons. Or something more complex; such as reducing the points shown by the .CSV or the data preparation functions in direct correlation with the current zooming level, so that Dygraphs (the graphing library) would only ever show x points at the same time.

The final decision was to downsample the full data set to a manageable size. This decision was made after looking at "Downsampling Time Series for Visual Representation" [43]. A recently published paper which looked at 3 different algorithms in order to downsample time-series specific data by separating the data points and treating each set of x points as buckets.

A lot of references had been made to this paper in online threads that proposed similar problems, almost always from a different graphing library or a different language altogether.

Someone had ported the algorithms described to the Node Package Manager, and therefore it was a simple `npm install git+https://github.com/pingec/downsample-lttb.git`. The "lttb" describes one of the three types of algorithms which is Largest Triangle Three Bucket, this algorithm takes into account the data points from the forward and backward buckets, instead of just the data points selected from the current bucket and therefore it is one that is better suited for larger trend-based data sets.

```
downsampler.processData(array, 20000);
//array in this scenario is the .csv
// with its first row (the headers) taken off
```

After some calculations it was understood that whilst the data would be downsampled, it would never be a misrepresentation of the car's run. If we downsample the data down to 20,000 points the graph would never not show the data for more than 1 second.

This might seem bad, however around 1 second is the worst case scenario, which is when the car is running an Endurance test.

Upon consultation with LUMotorsport's Chief Technical member, I was assured that the only time the data would need granularity to the 100Hz-level would be on small test, which are a few minutes. For the rest of the times, (ie, longer stints) the telemetry is used to detect information which doesn't a 100Hz frequency; for example driver's input, engine temperatures and suspension figures.

8.2.13 Version 22 - Downsampled data treating

With the data downsampled now it was possible to plot it back into a graph again, however because of the way the data is needed as input for Dygraphs there was some handling that needed to happen.

```
vflip(rotate(downsampler.processData(data, 20000)));
//rotate will turn the data from:
```

```
// a b c
// 1 2 3
//to:
// c 1
// b 2
// a 1
//uflip will turn the data from:
// 7 8 9
// 4 5 6
// 1 2 3
//to:
// 1 2 3
// 4 5 6
// 7 8 9
```

8.2.14 Version 23 - Gauges

After debugging the downsampling issue for quite some time I learned some more about the Dygraphs library and came to love some aspects of it. For example, it is possible for each point highlighting to do something by adding a function call in the options of the Dygraph chart.

```
//begin chart options after declaring chart location and data
highlightCallback: function(event, x, points, row) {
                console.log(x) // will print the x-value
              },
```

This really allowed me to shape the way I thought about creating gauges, as I know knew the kind of input I would be feeding them. After some research I stumbled across Gauge.js, a minimal library that has 2 styles of gauge (at this point in time most of the libraries for gauges I had found had tens of less professionally looking examples).

Implementing gauges and the text that comes with the gauge was really easy;

```
//begin chart options after declaring chart location and data
highlightCallback: function(event, x, points, row) {
                speedgauge.set(gaugereturn(row, graphs[74])); //[74] is speed
                speedgauge.setTextField(document.getElementById("speed-gauge-value"));
              },
```

After some styling I was able to have the gauges moving alongside the data with my cursor.

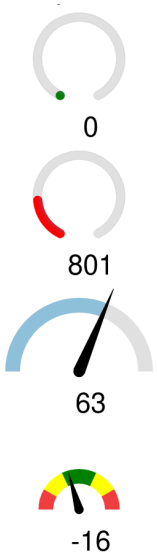


Figure 8.11: Gauges

8.2.15 Version 25 - Data Playback

In terms of data playback the approach was similar to the one of gauges, create a point which is affected as the data is moved.

```
function playback(){
      for(var i = 1; i < g.length; i++){
        if(playstate < 19999{ //19999 so it stops at the last point of the data
        // i know the last point because the data is always downsampled to 20000
          g[0].setSelection(playstate);
                // g[] is the array with all the graphs,
                // however since g is synchronized I just have to worry
                // about playing with the first graph and the rest will follow
}
```

After this I was also able to implement a speed setting and a pause setting by using Intervals, although I wasn't completely happy with the results as I have heard mixed reviews about using Intervals. However the Technical Lead seemed happy with the results and together we worked out the best 2 speeds to set it at.

```
setInterval(function(){playback()}, speed); //speed was 50 or 200.
```

8.2.16 Version 26 - Categorising Data Part 2

Some versions ago, in Version 14 I allowed the user to select where each of the channels should be displayed by offering a selection of HTML tab elements.

After showing this feature to the Technical Lead it was brought up that doing so would force the user to select the layout of a large amount of sensors every time the data was loaded.

As such, it was decided to port some of the requirements from this project to the Data Logger project and force its output .CSV to contain an extra row below the headers which contains the column where they should be displayed.

This method proved to be much quicker and intuitive than its first iteration;

```
for (var i = 0; i < graphs.length; i++){
        if(dataArr[1][i] == "drivercontrol"){
          g.push(new Dygraph(
            document.getElementById("graph" + i),
            // more graph creation parameters
```

This was then coupled with a faster way to render graphs and improve the overall performance, whereby each time a new tab was clicked only an 'impression' of switching tabs would happen. In the background the chart was being deleted and a new one was being rendered in its place with the appropiate channels.

```
function destroy(array){
  var container = document.getElementById(currentDiv);
  while (container.firstChild) {
    container.removeChild(container.lastChild);
  }
  for(var i = 1; i < array.length; i++){
    array[i].destroy();
  }
  array.length = 1;
}
// first item of the array is left on purpose
```

After this the Import Screen was no longer needed, however it was decided to be left there in case there was a future necessity in the development and process.

The UI was touched up slightly for the first time since the beginning and tabs where added along with a JS function which hid the panels not being displayed.

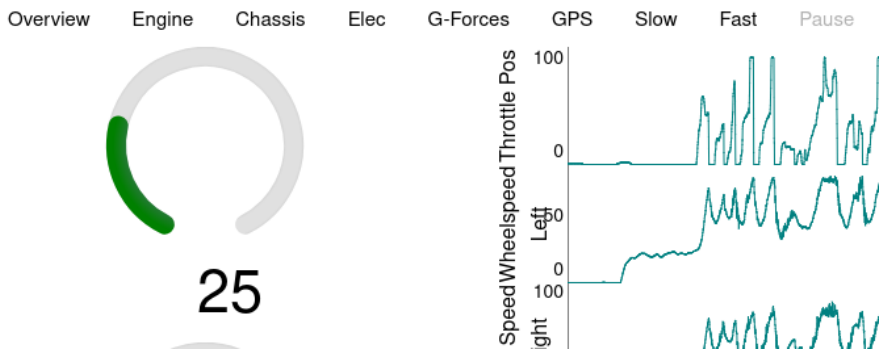


Figure 8.12: Channel Tabs

8.2.17 Version 27 - Per Lap Plotting; GPS data

Now it was time to tackle a new problem which would be split into different aspects. In order to plot a lap, the software needs to understand what a lap is. Therefore the fist thing to look at was the car's GPS readings, to understand how to register a lap.

After some sketching I decided to look at GPS Longitude and GPS Latitude only (leaving GPS Heading) and to try and establish a resolution/accuracy (by removing x amounts of data points) that would near 1-4 meters, big enough for the car to register every time it passed that points and small enough that the car doesn't think it's constantly passing the same spot.

```
var lat = 52.4932793;
var long = -1.1167698; ///static coordinates
var accuracy = 4;
var laps = [0];
for(var i = 0; i < cords[0].length; i++){
    if(cords[0][i].toFixed(accuracy) == lat.toFixed(accuracy)){
        if(cords[1][i].toFixed(accuracy) == long.toFixed(accuracy)){
            laps.push(i);
        }
    }
}
```

After that there were some repeating results and data which needed to get ready, so therefore some functions were created to treat the data, here is an example of the sort of functions that were used;

```
perlapcut(graphs[i], delta(removenear(laps, 50)));

function perlapcut(){
// this cuts the data right at the point of the longest lap and creates a 'new lap'
}
function delta(){
// returns the difference between laps after perlapcut has split them
}
function removenear(){
// this function cuts any repeating results from laps by 50
}
```

8.2.18 Version 29 - Per Lap Plotting; Times Table

After treating the information in Version 30 an HTML is dynamically created which contains the lap number, total time and lap time. These figures are generated using a combination of the functions described above - for example the total time does not use the delta() function. The resulting table also has a tickbox to show or hide the data.

8.2.19 Version 30 - Per Lap Plotting; Graph

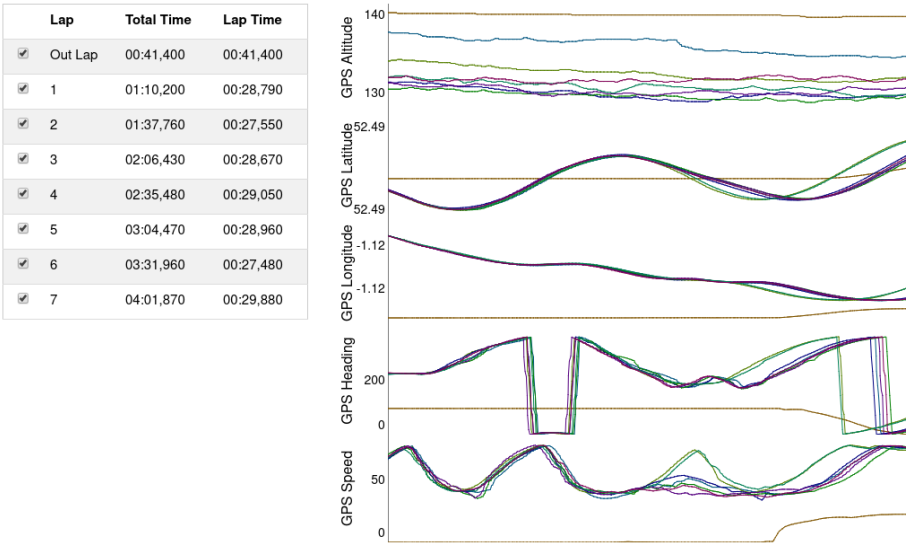
This part combines the GPS readings of Version 28 and the show/hide features from Version 29's table.

Thankfully Dyragphs syntax allows for a graph to have multiple y values by just prolonging the parameter of the Dygraph plotting function which references the data to be plotted. It therefore becomes as simple as;

```
g.push(new Dygraph(
  document.getElementById("graph" + i),
  vflip(rotate(perLapPlot)),
  // perLapPlot is:
  // [
```

```
// [time from 0 to the longest lap],
// [lap1 sensor reading A],
// [lap2 sensor reading A],
// [lapN sensor reading A]
//]
{animatedZooms: true, //more options
```

Combined with being able to plot more sensor readings here is the output;



	Lap	Total Time	Lap Time
☑	Out Lap	00:41,400	00:41,400
☑	1	01:10,200	00:28,790
☑	2	01:37,760	00:27,550
☑	3	02:06,430	00:28,670
☑	4	02:35,480	00:29,050
☑	5	03:04,470	00:28,960
☑	6	03:31,960	00:27,480
☑	7	04:01,870	00:29,880

Figure 8.13: Per Lap Page

8.2.20 Version 30 - Map

This section took some tries, but it was perfected and even features a different interaction model than any of the other charts whereby the user can span around the data by draggin the mouse (normally that would generate a zoom).

A Dygraph object was created and instead of it being plotted with Time vs Y, it was plotted with GPS Lat vs GPS Long allowing the track to be shown.

After that the map was plotted it had to respond, like the gauges, to both user highlights and the playback function. Thanks to the map also being a Dygraphs object it was easy to interact with it;

Figure 8.14: GPS Map

```
// this covers the user hovering
highlightCallback: function(event, x, points, row) {
                g[0].setSelection(row);
              }
// and this covers the playack hovering
g[0].setSelection(playstate);
```

8.2.21 Version 32 - Quick Plotter

The quick plotter is there for when team members need to quickly plot two values against each other, but don't need the extra computing capabilities of MATLAB. It appears as another tab and also follows the functionality of deleting all the charts as soon as the user switches to or away from the plotter.

HTML fields where used to create the dropdown menus which where then fed with all the headers from the .csv file with the help of JS;

```
for(var i = 0; i < title_array[0].length; i++) {
    var opt = title_array[0][i];
    var el = document.createElement("option");
    el.textContent = opt;
    el.value = opt;
    selectX.appendChild(el);
}
```

And then the values where temporarily saved when the user presses the plot button;

```
var quickplotselections = [];
var xaxis = selectX.options[selectX.selectedIndex].index;
var yaxis = selectY.options[selectY.selectedIndex].index;
quickplotselections.push(graphs[xaxis][1]);
quickplotselections.push(graphs[yaxis][1]);
```

8.2.22 Version 33 - Styling & Layout

As instructed in the specification document, the team would prefer to use some a darker theme as it will prevent eye strain. This section is one of the only ones in this particular project that has been affected by the COVID-19 regulation, as not being able able to have small and continuous feedback from LUMotorsport members has made UI development difficult.

Instead I have focused on getting a good base on what I consider to be my 'dark theme' theme of preference, and once the team is together make adjustments to the theme. According to an article [44], it is preferencial to avoid using a pure black wallpaper, instead opting for something like a dark grey. It is also recommendable to try and stay away from saturared colors by using more flat colors, which will not cause a 'highligther' effect on the screen.

The changes performed range from a simple HTML background change;

```
<body style="background-color: #121212">
```

to slightly more in-depth customization of the gauge elements - for example the steering angle;

```
var steeringops = {
  angle: 0,
  lineWidth: 0.05,
  radiusScale: 0.70,
  pointer: {
    length: 0.6,
    strokeWidth: 0.05,
    color: '#000000'
  },
  staticZones: [
    {strokeStyle: "#F03E3E", min: -90, max: -60},
    {strokeStyle: "yellow", min: -60, max: -25},
```

```
    {strokeStyle: "green", min: -25, max: 25},
    {strokeStyle: "yellow", min: 25, max: 60},
    {strokeStyle: "#F03E3E", min: 60, max: 90}
  ],
  staticLabels: {
    font: "10px sans-serif",  // Specifies font
    labels: [-90, -60, -25, 25, 60, 90],  // Print labels at these values
    color: "white",  // Optional: Label text color
  },
  pointer: {
    length: 0.6, // // Relative to gauge radius
    strokeWidth: 0.035, // The thickness
    color: 'white' // Fill color
  },
  limitMax: true,
  limitMin: true,
  strokeColor: '#E0E0E0',
  highDpiSupport: true
};
```

As well as accessing Dygraph's elements via their CSS;

```
.dygraph-xlabel {
  text-align: center;
  font-color: white;
}
```

After the color changes there were some layout considerations to be made. In Figure 5.12, which shows the gauges and the graph we can see that they are side by side - this use of space can prove the limiting. Instead I have chosen a light and easy to implement CSS framework that I have used in the past to be able to quickly position elements using a responsive column based layout.

The framework is W3.CSS, which has received some complaints in the past, for example by using classes as the way to access styling, by using `!important` more than 100 times in the code and by not having a GitHub page among some of the bigger complaints. However, and although I agree with most of the complaints, I find the framework to be quick to understand and very light, with only CSS and no JS (like for example Bootstrap). I did not want to complicate myself by having to learn another framework for the small CSS I had to do, and W3.CSS seemed like a good example.

Here is the result after some non exhaustive tinkering with some canvas styling and responsive layour which ensure the gauges and map are always given enough space. There are still some pieces which need the input of the LUMotorsport Technical Lead, such as how the graphs behave when there are plenty of them and whether scroll behaviour is encouraged or each individual graph should be reshaped to fit exactly one page.

Figure 8.15: Current styling and layout decisions

8.3 Black Box System Testing

When it comes to testing software there are two different areas. One area regards the knowledge the tester has about the code; there is White Box and Black Box testing. In White Box testing the user is aware of the path the program will take to give a results, whilst Black Box testing simply focuses on the input and the output - leaving the code to run without analysing it. I will be using Black Box testing, as I want my testing to be representative of the user's point of view. Furthermore Black Box testing will also allow the LUMotorsport team to test the program at the same level I tested it, avoiding developer bias and allowing both the client and myself to discuss the product along the same guidelines [46].

The second area regards the software's scope. There is Unit Testing; for individual code areas. Integration Testing; for the individual code to join and to work as one, System Testing; where the product is compared to the specification and Acceptance Testing; where the software is tested on a business environment and checked to see if it is ready for the final deployment.

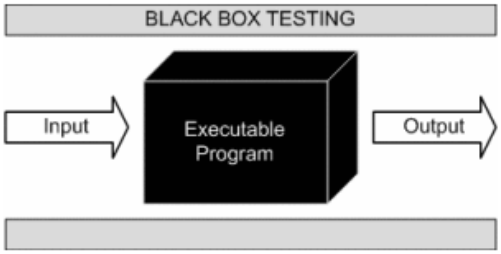


Figure 8.16: Visual representation of Black Box testing [46]

I will choose System testing, as I have a finished piece of software that is missing some minimal areas regarding the final implementation to the client's workplace due to the effects of COVID-19.

I will layout the software requirements for the software platform and assess one by one whether or not the criteria was met, whilst providing some comments regarding my overall experience of that particular task, if necessary.

This isn't the standard way of doing Black Box testing - normally there is a field for "Test Case No.", "Procedure to test" and "Expected Output". For my approach I will replace the case number by the enumerated list layout of the Requirements Document for the Telemetry Software, I will replace the "Procedure to Test" by a more informal approach of simply letting the item being tested use itself as its description and finally I will replace the "Expected Output" by color coded (green for pass, red for didn't pass, yellow for extraordinary cases). If the box is green I will explain the way it was achieved and if it is red I will go into a more formal approach as to why it did not work.

8.3.1 Testing List

1. Data Importation;
   (a) The software must accept a .CSV file

   > The software accepts .CSV thanks to the HTML input tag in the Import Screen

2. Data Preparation;
   (a) The dataset presented may have blank readings, the charting software shall still chart those readings.

Available by default thanks to Dygraphs library; "when Dygraphs encounters a missing value in a data series, it interprets this as a gap and draws it as such."

(b) Anything up to 40 minutes of data may be passed into the graphing software, the software must be able to handle that.

The `dataprep()` function turns the .CSV file into a downsampled version of itself with 20,000 data points. This means that the graphing software will be able to handle 40 minutes of data, with the only theoretical limitation to dataset length being the downsampler's ability to process data.

(c) Sensors must be separated into different categories, so that these can be plotted in different areas.

After two different ways of approaching the problem, the best solution found was the split them at the Data Logger and then make the graphing of each category look for the specific keyword.

3. Data Visualization;

(a) Once loaded the data must be responsive to a graph's area interaction, ie zooming, window resizing, panning.

Dygraph's canvas based library handles window resizing by re-rendering itself in a short amount of time which presents no noticeable lag to the user and thanks to Dygraph's customizeable interaction model; not only does zooming and resizing work, but they are also interchangable so that the mouse can affect each graph differently. As an extra to that, each Dygraph chart showing time-series data also has the option `animatedZooms` enabled for a smoother zooming experience.

(b) Time-series visualized data;

i. The user must be able to zoom and pan into the graph.

Dygraph's library handles this.

ii. There must be multi-channel graphs covering the Y-axis, all along the same time series.

This is achieved thanks to the passing of each graph as an element within one main array - `g.push(new Dygraph(...`

iii. The user must be able to overlap data from different times.

> A late development, but nonetheless an important one. This function comes as the second tab in the software "Per Lap Analysis" and is helped by the GPS data collected and treated.

iv. The cursor must act as a pointing device for the data, whereby the data point is highlighted.

> Default Dygraph behaviour allows for this.

v. The user must be able to add a 'point' in the data to reference later.

> Though Dygraphs allows code annotations as a built-in feature, I wasn't able to implement this. I picked the implementation time wrong and looked at this problem either too early (before learning about the graph updating features used for example in `g[i].setSelection(...)`) or too late, when time management required that I focused on other functionality aspects.

vi. The user must be able to filter out any overlapping data.

> Achieved through setting the built in `graph.setVisibility` by making a callback function to whenever a lap is hiden in the table `graphchange[i].setVisibility(parseInt(tickbox.id), tickbox.checked);`

(c) Non time-series visualized data;

i. The user must be able to plot any two channels against each other without the need for time to take part in the plotting.

This feature become it's own tab and is now called "Quick Plotter". Taking the input of an X value select HTML tag and a Y value select HTML tag which are filled with every header from the .csv file.

However, Dygraphs does not allow for Scatter Plots and therefore sometimes the data might not appear as the user desires - typically changing the X and Y values around will help the chart visualization, however it isn't perfect.

If I was to do this again I would make sure to use another library which was orientated towards more graph charts rather than pure performance. A further downsampled data set being fed into a Chart.js instance is an idea that was thrown around however it never came to fruition due to time constraints.

(d) Map and lap splits;

i. The software must find the point when the car starts a new lap.

This is done thanks to the program's interaction with the GPS data similarly to 3(b)iii. The code will reduce the coordinate's resolution to a span of 1 to 4 meters and allow for the car to realize when it is 'touching' a path that it already has gone through. This is then registered as a lap.

ii. Laps must be plotted on a table which shows;

A. Lap number, including the Out Lap.

Achieved, with lap number being the index of the `laps` array.

B. Total run time.

Achieved, by showing the non-delta'd time.

C. Lap time.

Achieved, by showing the delta'd time.

iii. There must be a map of the circuit.

This is achieved thanks to the passing of each graph as an element within one main array - `g.push(new Dygraph(...`

iv. The user must be able to hover the data and see the car's position in the circuit.

Achieved by using the `g[0].setSelection(row);` option, where `g[0]` always represents the map element.

(e) Gauges;

i. There must be gauges for;

A. Throttle Input.

Achieved, by using the gauge.js library and a text tag that showed the value as a text parameter

B. Brake Pressure.

Achieved, by using the gauge.js library and a text tag that showed the value as a text parameter

C. Steering Angle.

Achieved, by using the gauge.js library and a text tag that showed the value as a text parameter

D. Vehicle Speed

Achieved, by using the gauge.js library and a text tag that showed the value as a text parameter

ii. The gauges must, like the map, change accordingly to the user's current hover position over the graphs.

The gauges used `speedgauge.set(gaugereturn(row, graphs[x]));` from within the `highlightCallback: function(event, x, points, row)` function in order to, like the map, change to the user's mouse position.

(f) Data Playback;

i. When the user chooses to, the program must be able to playback the data in the graphs.

This was achieved with the `playback()` function which, using a time interval, moves the graph's pointer one point ahead `g[i].setSelection(playstate)` until the limit of 20,000 data points.

ii. The data must, as with user mouse hovering, display a pointing device showing the values at that specific point.

This was built in, as Dygraph interprets a `setSelection()` function as a mouse interaction by the user.

iii. The speed the playback happens at must be selectable from a range of options.

Achieved thanks to the `playbackfunction()`'s interchanging interval setting.

iv. The user will be able to start the data playing back at any time point.

This was not achieved. That said, this was a time limitation and not a complexity one. The way to do this would only take some lines of code an an HTML text input element. The `var playback` would then be set to the user's request.

v. The user will be able to stop the data playback.

Achieved, thanks to the `clearInterval()` function which is mapped to a stop button in the data playback area.

vi. The user will be able to take over graph hovering or zooming, and then let the data playback carry on its work.

This was built in, as Dygraph interprets a `setSelection()` function as a mouse interaction by the user.

4. User Interface;

(a) A dark theme must be implemented across the system in order to reduce eye strain.

As seen in the last chapter of Implementation, a combination of JS styling for the gauges, CSS styling for dygraphs and basic CSS for the HTML page gave the page a 'dark-like' theme, pending changes by the LUMotorsport team.

(b) The user must be able to navigate the software intuitively.

There are tabs to navigate through sections.

(c) There will be no new windows opening to show different graphs, everything must be kept on the same screen.

Everything is accessible from the main page, in exception for the Import Screen which was left there as per the request of LUMotorsport's Chief Technical member, in case of future upgrades to the data handling.

5. Technical;
   (a) The solution must be compatible with Windows, MacOS and Linux.

   Written in Electron, software cross-compatibility was a high priority for choosing the right development environment.

   (b) The solution must be ready to be upgraded;
      i. By using open source libraries.

      Check.

      ii. By taking a modular approach to its separate parts.

      Functions were used throughout the code and there is a clear modularity between each Dygraph element.

8.3.2 Testing Results

Table 8.1: Split of test results as a form of performance

Passed Tests	Failed Test	Extraordinary Tests
33	2	1

8.4 Screenshots

Figure 8.17: Time-series Plot

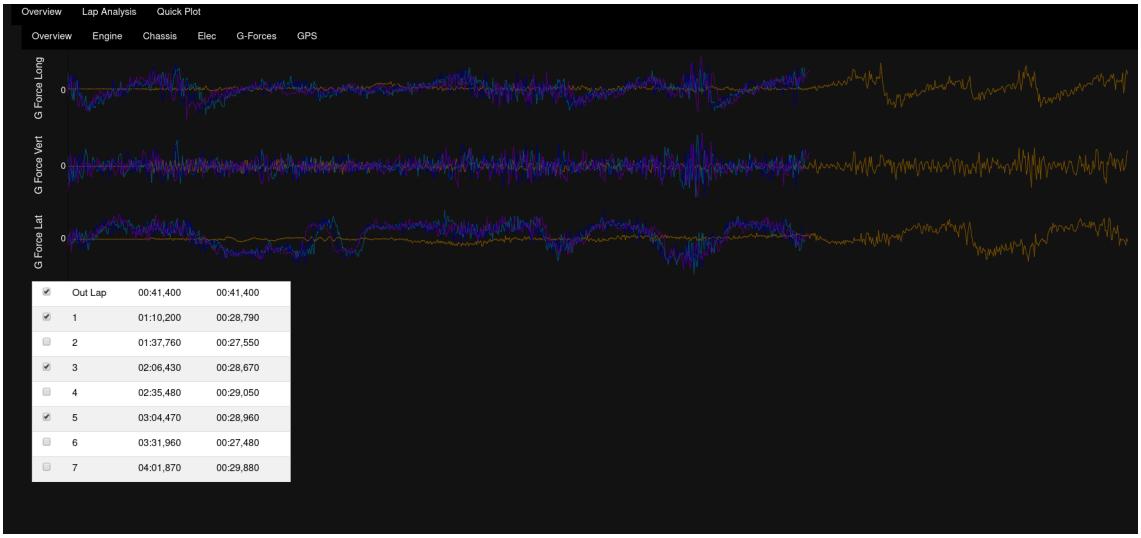


Figure 8.18: Per Lap Analysis

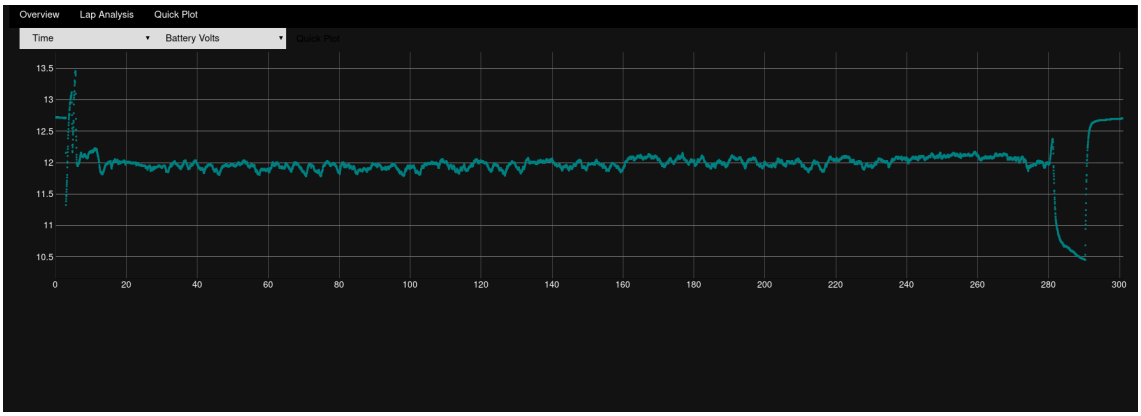


Figure 8.19: Quick Plotter

9. Conclusion & Final Results

9.1 Estimation of Costs Saved

Apologies in advance as this section was meant to be larger, however due to the lockdown the team haven't been able to meet for this particular meeting. In this meeting the Electronics, Driver Controls, High Voltage and Software Development (me) teams would have sat down and looked through last year's itemized cost report, discounting every part that has been replaced.

From my estimates however I believe that through the work to implement the CAN network, the custom data logger and the graph visualization software the team have saved the team around $3000 to $5000 in the following years. That is an increble success.

9.2 The Output Produced

Overall, I am happy with the output of this project. It was a tough call to try and tackle 4 individual projects, even though the hardware-based projects were smaller than the bigger telemetry software, it was still incredibly challenging to do this project.

I wanted to pick something I was passionate about, something that I would be proud of it at end and something that would help LUMotorsport, who for my 3 years at my university have been one of my only steady sources of fun and challenges. Being already part of the team and understanding the culture, the processes and the output expected definitely made it easier, as had I not known the team before I would have been scared to build up the confidence to tackle the problems I wanted to take.

9.3 Improvements & Upgrades

There are a large number of improvements that I would consider these 4 projects need - and most of these fall down to not having enough time with the tools. In part this is my fault, as time management has always been a struggle. However, another even bigger part has been the COVID lockdown and other circumstances of the year which made it tough (or impossible) to go to the workshop and work on the physical projects.

- For the telemetry software I would have liked the complete all the tasks required, and furthermore I would like to combine a wider range of charting libraries - so that the team does not rely on another platform (MATLAB) continuously. Furthermore I would like

to work more on the layout, as I am a big believer that improving the user interface and styling of a piece of software can make the difference. An example of this would be graph line coloring based on the attribute.

As an additional point, I think a good improvement (though much more complicated) would be doing the software in RStudio - even though it wasn't chosen at the end, it was the software which impressed me the most and I will look at it in the near future.

- The CAN network and data logger suffered delays, despite the lockdown which eventually cancelled the project. The delays came in the form of communication errors, rather than technical errors. Perhaps the biggest communication mistake was the delay and uncertainty on the rule clarification - which changed some of the project's requirements with the addition of the safety regulated sensor readings.
- And for the live telemetry, I believe with an antenna that can deliver WiFi at long range - the car could have a functioning critical information MATLAB based plotting system, which the team could tweak depending on the test that needs performing.

I hope that special consideration can be taken with my responses to the above points, as it is hard to calculate the improvements necessary on hardware and software that was only partly finished.

9.4 What Went Badly

Again, this was mostly time management. Throughout the year I have had personal issues which have kept me away from university for almost a month - and even before those issues appeared I had an issue with my student status which made me start the Final Year Project a bit behind schedule.

Without turning to those areas however, I still believe that my time management was poor in some areas. I struggled with keeping motivated throughout the development process, as sometimes finding a bug would take me hours of research to fix - something which I found demoralising.

The worst part of the project was the Performance issues with the Telemetry Software. It was something which was hard to get over, as I had spent some time implementing changes, and even had gone version ahead - thinking that the data rendering would be flawless at higher sizes.

Having to see the data load for 60 seconds before showing, and knowing that I had to rewrite several iterations again was challenging.

9.5 Learning Outcomes

Overall, I have learned a great deal;

- Electron; I had kept my eye on this platform for about two years, seeing it slowly grow and it was great to finally be able to use it. Most of the team have MacBook's which MoTeC does not work with and it was great to be able to make a piece of software cross-platform.
- Electron's Alternatives; MATLAB and RStudio are fantastic pieces of software and they have given a new understanding of the Science-related Computer Science field.
- Electronics; It has been great to be able to use Arduino's and learn about the communication protocols in detail. I am confident this will not be my last electronics project.
- Communication skills; Having weekly and fortnightly meetings where I need to give feedback, talking to the "client" and drafting a specification documents and having to quickly react to the changing environment have been great ways to get some extra experience before leaving univeristy.

10. Appendix A - Telemetry Dashboard Software

10.1 app.js

```
process.env['ELECTRON_DISABLE_SECURITY_WARNINGS'] = 'true';
const fs = require('fs');
const parse = require('csv-parse');
const path = require('path');
const ipc = require('electron').ipcRenderer;
const BrowserWindow = require('electron').remote.BrowserWindow;
var downsampler = require("downsample-lttb");
const rotate = require('2d-array-rotation').rotate270;
const vflip = require('2d-array-rotation').vflip;
const openSecondWindowButton = document.getElementById('open-second-window');
const container1button = document.getElementById('container1button');
const container2button = document.getElementById('container2button');
const container3button = document.getElementById('container3button');
const container4button = document.getElementById('container4button');
const container5button = document.getElementById('container5button');
const container6button = document.getElementById('container6button');
const container1lapbutton = document.getElementById('container1lapbutton');
const container2lapbutton = document.getElementById('container2lapbutton');
const container3lapbutton = document.getElementById('container3lapbutton');
const container4lapbutton = document.getElementById('container4lapbutton');
const container5lapbutton = document.getElementById('container5lapbutton');
const container6lapbutton = document.getElementById('container6lapbutton');
const playbutton = document.getElementById('playbutton');
const pausebutton = document.getElementById('pausebutton');
const speedbutton = document.getElementById('speedbutton');
const quickplotbutton = document.getElementById('quickplotbutton');
const extrastab = document.getElementById('extrastab');
var selectX = document.getElementById("selectX");
var selectY = document.getElementById("selectY");
var perlap = document.getElementById("perlap");
var gaugediv = document.getElementById("gaugediv");
var laptablediv = document.getElementById("laptablediv");
var table = document.getElementById("table");
var showlap = document.getElementById("showlap");

var speedopts = {
  angle: 0,
  lineWidth: 0.2, // The line thickness
```

```
  radiusScale: 0.5, // Relative radius
  pointer: {
    length: 0.71, // // Relative to gauge radius
    strokeWidth: 0.062, // The thickness
    color: 'white' // Fill color
  },
  limitMax: false,     // If false, max value increases automatically if value > maxValue
  limitMin: false,     // If true, the min value of the gauge will be fixed
  colorStart: '#6FADCF',   // Colors
  colorStop: '#8FC0DA',    // just experiment with them
  strokeColor: '#E0E0E0',  // to see which ones work best for you
  generateGradient: true,
  highDpiSupport: true,     // High resolution support


};
var brakeopts = {
  angle: 0.24, // The span of the gauge arc
  lineWidth: 0.05, // The line thickness
  radiusScale: 0.75, // Relative radius
  limitMax: false,     // If false, max value increases automatically if value > maxValue
  limitMin: false,     // If true, the min value of the gauge will be fixed
  colorStart: '#A01A27',   // Colors
  colorStop: '#FF0A13',    // just experiment with them
  strokeColor: '#EEEEEE',  // to see which ones work best for you
  generateGradient: true,
  highDpiSupport: true,     // High resolution support
};
var throttleopts = {
  angle: 0.24, // The span of the gauge arc
  lineWidth: 0.05, // The line thickness
  radiusScale: 0.75, // Relative radius
  limitMax: false,     // If false, max value increases automatically if value > maxValue
  limitMin: false,     // If true, the min value of the gauge will be fixed
  colorStart: '#12A038',   // Colors
  colorStop: '#3DDB25',    // just experiment with them
  strokeColor: '#EEEEEE',  // to see which ones work best for you
  generateGradient: true,
  highDpiSupport: true     // High resolution support
};
var steeringops = {
```

```
  angle: 0,
  lineWidth: 0.05,
  radiusScale: 0.70,
  pointer: {
    length: 0.6,
    strokeWidth: 0.05,
    color: '#000000'
  },
  staticZones: [
    {strokeStyle: "#F03E3E", min: -90, max: -60},
    {strokeStyle: "yellow", min: -60, max: -25},
    {strokeStyle: "green", min: -25, max: 25},
    {strokeStyle: "yellow", min: 25, max: 60},
    {strokeStyle: "#F03E3E", min: 60, max: 90}
  ],
  staticLabels: {
    font: "10px sans-serif",  // Specifies font
    labels: [-90, -60, -25, 25, 60, 90],  // Print labels at these values
    color: "white",  // Optional: Label text color
  },
  pointer: {
    length: 0.6, // // Relative to gauge radius
    strokeWidth: 0.035, // The thickness
    color: 'white' // Fill color
  },
  limitMax: true,
  limitMin: true,
  strokeColor: '#E0E0E0',
  highDpiSupport: true
};

var throttlegaugediv = document.getElementById('throttlegauge'); // your canvas element
var throttlegauge = new Donut(throttlegaugediv).setOptions(throttleopts); // create sexy
throttlegauge.maxValue = 100;
throttlegauge.animationSpeed = 128;
throttlegauge.set(0);
var brakegaugediv = document.getElementById('brakegauge'); // your canvas element
var brakegauge = new Donut(brakegaugediv).setOptions(brakeopts); // create sexy gauge!
brakegauge.maxValue = 5000;
brakegauge.animationSpeed = 128;
brakegauge.setMinValue(0);
```

```
brakegauge.set(0);
var speedgaugediv = document.getElementById('speedgauge'); // your canvas element
var speedgauge = new Gauge(speedgaugediv).setOptions(speedopts); // create sexy gauge!
speedgauge.maxValue = 100;
speedgauge.animationSpeed = 128;
speedgauge.set(0);
var steeringgaugediv = document.getElementById('steeringgauge'); // your canvas element
var steeringgauge = new Gauge(steeringgaugediv).setOptions(steeringops); // create sexy g
steeringgauge.maxValue = 90;
steeringgauge.animationSpeed = 128;
steeringgauge.setMinValue(-90);
steeringgauge.set(0);


var dataArr = [];
var graphs = [];
var g = [];
var inputFile='';
var tabSelector = '';
var labels = [];
var dataArrNumber = [];
var currentDiv = '';
var playstate = 0;
var idVar;
var title_array = [];
var cords = [];

ipc.on('messageFromMain', (event, message) => {
  inputFile = JSON.parse(message).data;
  tabSelector = JSON.parse(message).column;
  dataprep();
});

function destroy(array){
  var container = document.getElementById(currentDiv);
  while (container.firstChild) {
    container.removeChild(container.lastChild);
  }
  for(var i = 1; i < array.length; i++){
    array[i].destroy();
  }
```

```
  array.length = 1;
}

function removenear(array, diff){
  var output = [];
  for(var i = 0; i < array.length; i++){
    if(array[i] - array[i-1] > diff){
      output.push(array[i]);
    }
  }
  return output
}

function perlapcut(x, cut){
  x = x.slice(0);
  var splicer = x[1].slice();
  var output = [];
  for(var i = 0; i < cut.length; i++){
    output.push(splicer.splice(0, (cut[i])));
    //x.shift(cut[i]);
  }
  return output;
};

function delta(x){
  var b = [x[0]];
  for(var i = 0; i < x.length-1; i++){
    b.push(x[i+1] - x[i]);
  }
  //b.pop();
  return b;
}

function gaugereturn(value, array){
  return array[1][value];
}

function longest(array){
  var max = 0
  for(var i = 0; i < array.length; i++){
    if(array[i].length > max){
```

```
      max = array[i].length;
    }
  }
  return max
}

function addRow(lap, timelap, timecumu, graphchange){
  var x=document.getElementById('table').insertRow(-1);
  var show = x.insertCell(0);
  var lapNo = x.insertCell(1);
  var cumuTime = x.insertCell(2);
  var lapTime = x.insertCell(3);
  var tickbox = document.createElement("input");
  tickbox.setAttribute("type", "checkbox");
  tickbox.setAttribute("id", lap);
  tickbox.setAttribute("class", "tick");
  tickbox.setAttribute("checked", true);
  tickbox.addEventListener('change', function change() {
    for(var i = 1; i < graphchange.length; i++){
      graphchange[i].setVisibility(parseInt(tickbox.id), tickbox.checked);
    }
  });
  show.appendChild(tickbox);
  if(lap == 0){
    lapNo.innerHTML="Out Lap";
  }
  else{
    lapNo.innerHTML=lap;
  }
  cumuTime.innerHTML=timecumu;
  lapTime.innerHTML=timelap;
};

function sec2time(timeInSeconds) {
  var pad = function(num, size) { return ('000' + num).slice(size * -1); },
  time = parseFloat(timeInSeconds).toFixed(3),
  hours = Math.floor(time / 60 / 60),
  minutes = Math.floor(time / 60) % 60,
  seconds = Math.floor(time - minutes * 60),
  milliseconds = time.slice(-3);
  return pad(minutes, 2) + ':' + pad(seconds, 2) + ',' + pad(milliseconds, 3);
```

```
}

function dataprep(){
  var parser = parse({delimiter: ','}, function (data) {
    for(var i = 3; i < dataArr.length; i++){
      dataArrNumber.push(dataArr[i].map(Number));
    }
    var rotated = vflip(rotate(downsampler.processData(dataArrNumber, 10000))); // TODO:
    for (var i = 0; i < rotated.length; i++){
      graphs.push([rotated[0], rotated[i]]);
    }
    //console.log(graphs[1][1]);
    title_array.push(dataArr[0]);
    cords.push(rotated[69]);
    cords.push(rotated[70]);
    g.push(new Dygraph(
      document.getElementById("map_container"),
      vflip(rotate([rotated[70], rotated[69]])),
      {animatedZooms: true,
        interactionModel : {
          'mousedown' : downV3,
          'mousemove' : moveV3,
          'mouseup' : upV3,
          'click' : clickV3,
          'dblclick' : dblClickV3,
          'mousewheel' : scrollV3
        },
        height: 400,
        width: 400,
        highlightCircleSize: 8,
        drawGrid: false,
        drawAxis: false,
        legend: 'never'}
      ));
      for (var i = 0; i < graphs.length; i++){
        if(dataArr[1][i] == "drivercontrol"){
          document.getElementById('container1').innerHTML+=('<div id="graph' + i + '" sty
        }
      }
      for (var i = 0; i < graphs.length; i++){
        if(dataArr[1][i] == "drivercontrol"){
```

```
      //console.log(graphs[]);
      g.push(new Dygraph(
        document.getElementById("graph" + i),
        vflip(rotate(graphs[i])),
        {animatedZooms: true,
          labels: [dataArr[0][0], dataArr[0][i]],
          ylabel: dataArr[0][i],
          highlightCallback: function(event, x, points, row) {
            g[0].setSelection(row);
            speedgauge.set(gaugereturn(row, graphs[74]));
            steeringgauge.set(gaugereturn(row, graphs[67]));
            brakegauge.set(gaugereturn(row, graphs[28]));
            throttlegauge.set(gaugereturn(row, graphs[5]));
            speedgauge.setTextField(document.getElementById("speed-gauge-value"));
            steeringgauge.setTextField(document.getElementById("steering-gauge-value"
            throttlegauge.setTextField(document.getElementById("throttle-gauge-value"
            brakegauge.setTextField(document.getElementById("brake-gauge-value"));
          },
          //axes: {x: {drawAxis: false, valueFormatter: function(col) {console.log('c
          axes: {x: {drawAxis: false}},
          drawGrid: false,
          plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})],
          legend: 'follow'}
        ));
      }
    }
    Dygraph.synchronize(g.slice(1), {
      zoom: true,
      selection: true,
      range: false
    });
    currentDiv = "container1"
  });
  fs.createReadStream(inputFile)
  .pipe(parser)
  .on('data', (data) => {dataArr.push(data.map(function(x){return x;}))});
};



playbutton.addEventListener('click', event => {
```

```
  clearInterval(idVar);
  playbutton.disabled = true;
  pausebutton.disabled = false;
  speedbutton.disabled = false;
  idVar = setInterval(function(){playback()}, 300);
});
speedbutton.addEventListener('click', event => {
  clearInterval(idVar);
  speedbutton.disabled = true;
  playbutton.disabled = false;
  pausebutton.disabled = false;
  idVar = setInterval(function(){playback()}, 50);
});
pausebutton.addEventListener('click', event => {
  clearInterval(idVar);
  //playstate = 0;
  playbutton.disabled = false;
  speedbutton.disabled = false;
  pausebutton.disabled = true;
});
function playback(){
  for(var i = 1; i < g.length; i++){
    if(playstate < 9999){
      g[i].setSelection(playstate);
      g[0].setSelection(playstate);
      speedgauge.set(gaugereturn(playstate, graphs[74]));
      steeringgauge.set(gaugereturn(playstate, graphs[67]));
      brakegauge.set(gaugereturn(playstate, graphs[28]));
      throttlegauge.set(gaugereturn(playstate, graphs[5]));
      speedgauge.setTextField(document.getElementById("speed-gauge-value"));
      steeringgauge.setTextField(document.getElementById("steering-gauge-value"));
      throttlegauge.setTextField(document.getElementById("throttle-gauge-value"));
      brakegauge.setTextField(document.getElementById("brake-gauge-value"));
      playstate = playstate + 1;
    }
    else{
      clearInterval(idVar);
      playbutton.disabled = false;
      speedbutton.disabled = false;
      pausebutton.disabled = true;
      playstate = 0;
```

```
    }
  };
};



container1button.addEventListener('click', event => {
  console.log(g);
  destroy(g);
  console.log(g);

  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "drivercontrol"){
      document.getElementById('container1').innerHTML+=('<div id="graph' + i + '" sty
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "drivercontrol"){
      //console.log(graphs[]);
      g.push(new Dygraph(
        document.getElementById("graph" + i),
        vflip(rotate(graphs[i])),
        {animatedZooms: true,
          labels: [dataArr[0][0], dataArr[0][i]],
          ylabel: dataArr[0][i],
          highlightCallback: function(event, x, points, row) {
            g[0].setSelection(row);
            speedgauge.set(gaugereturn(row, graphs[74]));
            steeringgauge.set(gaugereturn(row, graphs[67]));
            brakegauge.set(gaugereturn(row, graphs[28]));
            throttlegauge.set(gaugereturn(row, graphs[5]));
            speedgauge.setTextField(document.getElementById("speed-gauge-value"));
            steeringgauge.setTextField(document.getElementById("steering-gauge-value'
            throttlegauge.setTextField(document.getElementById("throttle-gauge-value'
            brakegauge.setTextField(document.getElementById("brake-gauge-value"));
          },
          //axes: {x: {drawAxis: false, valueFormatter: function(col) {console.log('
          axes: {x: {drawAxis: false}},
          drawGrid: false,
          plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})],
          legend: 'follow'}
```

```
      ));
    }
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
  });
  currentDiv = "container1"
});



container2button.addEventListener('click', event => {
  destroy(g);
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "engine"){
      document.getElementById('container2').innerHTML+=('<div id="graph' + i + '" s
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "engine"){
      g.push(new Dygraph(
        document.getElementById("graph" + i),
        vflip(rotate(graphs[i])),
        {animatedZooms: true,
          labels: [dataArr[0][0], dataArr[0][i]],
          ylabel: dataArr[0][i],
          highlightCallback: function(event, x, points, row) {
            g[0].setSelection(row);
            speedgauge.set(gaugereturn(row, graphs[74]));
            steeringgauge.set(gaugereturn(row, graphs[67]));
            brakegauge.set(gaugereturn(row, graphs[28]));
            throttlegauge.set(gaugereturn(row, graphs[5]));
            speedgauge.setTextField(document.getElementById("speed-gauge-value"));
            steeringgauge.setTextField(document.getElementById("steering-gauge-valu
            throttlegauge.setTextField(document.getElementById("throttle-gauge-valu
            brakegauge.setTextField(document.getElementById("brake-gauge-value"));
          },
          //axes: {x: {drawAxis: false, valueFormatter: function(col) {console.log(
          axes: {x: {drawAxis: false}},
```

```
        drawGrid: false,
        plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})],
        legend: 'follow'}
      ));
    }
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
  });
  currentDiv = "container2"
});



container3button.addEventListener('click', event => {
  destroy(g);
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "chassis"){
      document.getElementById('container3').innerHTML+=('<div id="graph' + i + ''
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "chassis"){
      g.push(new Dygraph(
        document.getElementById("graph" + i),
        vflip(rotate(graphs[i])),
        {animatedZooms: true,
          labels: [dataArr[0][0], dataArr[0][i]],
          ylabel: dataArr[0][i],
          highlightCallback: function(event, x, points, row) {
            g[0].setSelection(row);
            speedgauge.set(gaugereturn(row, graphs[74]));
            steeringgauge.set(gaugereturn(row, graphs[67]));
            brakegauge.set(gaugereturn(row, graphs[28]));
            throttlegauge.set(gaugereturn(row, graphs[5]));
            speedgauge.setTextField(document.getElementById("speed-gauge-value"))
            steeringgauge.setTextField(document.getElementById("steering-gauge-va
            throttlegauge.setTextField(document.getElementById("throttle-gauge-va
            brakegauge.setTextField(document.getElementById("brake-gauge-value"))
```

```
        },
        //axes: {x: {drawAxis: false, valueFormatter: function(col) {console.lo
        axes: {x: {drawAxis: false}},
        drawGrid: false,
        plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})],
        legend: 'follow'}
      ));
    }
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
  });
  currentDiv = "container3"
});



container4button.addEventListener('click', event => {
  destroy(g);
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "elec"){
      document.getElementById('container4').innerHTML+=('<div id="graph' + i +
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "elec"){
      g.push(new Dygraph(
        document.getElementById("graph" + i),
        vflip(rotate(graphs[i])),
        {animatedZooms: true,
          labels: [dataArr[0][0], dataArr[0][i]],
          ylabel: dataArr[0][i],
          highlightCallback: function(event, x, points, row) {
            g[0].setSelection(row);
            speedgauge.set(gaugereturn(row, graphs[74]));
            steeringgauge.set(gaugereturn(row, graphs[67]));
            brakegauge.set(gaugereturn(row, graphs[28]));
            throttlegauge.set(gaugereturn(row, graphs[5]));
            speedgauge.setTextField(document.getElementById("speed-gauge-value"
```

```
            steeringgauge.setTextField(document.getElementById("steering-gauge-
            throttlegauge.setTextField(document.getElementById("throttle-gauge-
            brakegauge.setTextField(document.getElementById("brake-gauge-value'
          },
          //axes: {x: {drawAxis: false, valueFormatter: function(col) {console.
          axes: {x: {drawAxis: false}},
          drawGrid: false,
          plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})],
          legend: 'follow'}
        ));
      }
    }
    Dygraph.synchronize(g.slice(1), {
      zoom: true,
      selection: true,
      range: false
    });
    currentDiv = "container4"
});



container5button.addEventListener('click', event => {
    destroy(g);
    for (var i = 0; i < graphs.length; i++){
      if(dataArr[1][i] == "gforce"){
        document.getElementById('container5').innerHTML+=('<div id="graph' + i
      }
    }
    for (var i = 0; i < graphs.length; i++){
      if(dataArr[1][i] == "gforce"){
        g.push(new Dygraph(
          document.getElementById("graph" + i),
          vflip(rotate(graphs[i])),
          {animatedZooms: true,
            labels: [dataArr[0][0], dataArr[0][i]],
            ylabel: dataArr[0][i],
            highlightCallback: function(event, x, points, row) {
              g[0].setSelection(row);
              speedgauge.set(gaugereturn(row, graphs[74]));
              steeringgauge.set(gaugereturn(row, graphs[67]));
```

```
            brakegauge.set(gaugereturn(row, graphs[28]));
            throttlegauge.set(gaugereturn(row, graphs[5]));
            speedgauge.setTextField(document.getElementById("speed-gauge-valu
            steeringgauge.setTextField(document.getElementById("steering-gaug
            throttlegauge.setTextField(document.getElementById("throttle-gaug
            brakegauge.setTextField(document.getElementById("brake-gauge-valu
          },
          //axes: {x: {drawAxis: false, valueFormatter: function(col) {consol
          axes: {x: {drawAxis: false}},
          drawGrid: false,
          plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})],
          legend: 'follow'}
        ));
      }
    }
    Dygraph.synchronize(g.slice(1), {
      zoom: true,
      selection: true,
      range: false
    });
    currentDiv = "container5"
});



container6button.addEventListener('click', event => {
    destroy(g);
    for (var i = 0; i < graphs.length; i++){
      if(dataArr[1][i] == "gps"){
        document.getElementById('container6').innerHTML+=('<div id="graph' +
      }
    }
    for (var i = 0; i < graphs.length; i++){
      if(dataArr[1][i] == "gps"){
        g.push(new Dygraph(
          document.getElementById("graph" + i),
          vflip(rotate(graphs[i])),
          {animatedZooms: true,
            labels: [dataArr[0][0], dataArr[0][i]],
            ylabel: dataArr[0][i],
            highlightCallback: function(event, x, points, row) {
```

```
            g[0].setSelection(row);
            speedgauge.set(gaugereturn(row, graphs[74]));
            steeringgauge.set(gaugereturn(row, graphs[67]));
            brakegauge.set(gaugereturn(row, graphs[28]));
            throttlegauge.set(gaugereturn(row, graphs[5]));
            speedgauge.setTextField(document.getElementById("speed-gauge-va
            steeringgauge.setTextField(document.getElementById("steering-ga
            throttlegauge.setTextField(document.getElementById("throttle-ga
            brakegauge.setTextField(document.getElementById("brake-gauge-va
          },
          //axes: {x: {drawAxis: false, valueFormatter: function(col) {cons
          axes: {x: {drawAxis: false}},
          drawGrid: false,
          plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})]
          legend: 'follow'}
        ));
      }
    }
    Dygraph.synchronize(g.slice(1), {
      zoom: true,
      selection: true,
      range: false
    });
    currentDiv = "container6"
});

extrastab.addEventListener('click', event => {
    destroy(g);
    console.log(title_array[0][1]);
    for(var i = 0; i < title_array[0].length; i++) {
      var opt = title_array[0][i];
      var el = document.createElement("option");
      el.textContent = opt;
      el.value = opt;
      selectX.appendChild(el);
    }
    for(var i = 0; i < title_array[0].length; i++) {
      var opt = title_array[0][i];
      var el = document.createElement("option");
      el.textContent = opt;
      el.value = opt;
```

```
    selectY.appendChild(el);
  }
});



var quickplots = 0;
quickplotbutton.addEventListener('click', event => {
  quickplots++;
  var quickplotselections = [];
  var xaxis = selectX.options[selectX.selectedIndex].index;
  var yaxis = selectY.options[selectY.selectedIndex].index;
  quickplotselections.push(graphs[xaxis][1]);
  quickplotselections.push(graphs[yaxis][1]);
  console.log(quickplotselections);
  document.getElementById('quickplot_container').innerHTML+=('<div id="qu
  new Dygraph(
    document.getElementById('quickplot'+quickplots),
    vflip(rotate(quickplotselections)),
    {animatedZooms: true,
      drawPoints: true,
      interactionModel : {
        'mousedown' : downV3,
        'mousemove' : moveV3,
        'mouseup' : upV3,
        'click' : clickV3,
        'dblclick' : dblClickV3,
        'mousewheel' : scrollV3
      },
      //labels: [dataArr[0][0], dataArr[0][i]],
      //ylabel: dataArr[0][i],
      highlightCallback: function(event, x, points, row) {
        g[0].setSelection(row);
        speedgauge.set(gaugereturn(row, graphs[74]));
        steeringgauge.set(gaugereturn(row, graphs[67]));
        brakegauge.set(gaugereturn(row, graphs[28]));
        throttlegauge.set(gaugereturn(row, graphs[5]));
        speedgauge.setTextField(document.getElementById("speed-gauge-valu
        steeringgauge.setTextField(document.getElementById("steering-gaug
        throttlegauge.setTextField(document.getElementById("throttle-gaug
```

```
      brakegauge.setTextField(document.getElementById("brake-gauge-valu
    },
    //axes: {x: {drawAxis: false, valueFormatter: function(col) {consol
    //axes: {x: {drawAxis: false}},
    //drawGrid: false,
    strokeWidth: 0,
    plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical"})],
    legend: 'follow'}
  );
  currentDiv = "quickplot_container";
});



container1lapbutton.addEventListener('click', event => {
  table.innerHTML = "";
  destroy(g);
  var lat = 52.4932793;
  var long = -1.1167698;
  var head = 90.02;
  var accuracy = 4;
  var laps = [0];
  for(var i = 0; i < cords[0].length; i++){
    if(cords[0][i].toFixed(accuracy) == lat.toFixed(accuracy)){
      if(cords[1][i].toFixed(accuracy) == long.toFixed(accuracy)){
        laps.push(i);
      }
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "drivercontrol"){
      document.getElementById("container1lap").innerHTML+=('<div id="gr
    }
  }
  var lapsArray = removenear(laps, 50);
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "drivercontrol"){
      perLapPlot = perlapcut(graphs[i], delta(removenear(laps, 50)));
      perLapPlot.unshift(graphs[0][0].slice());
      console.log(graphs[70]);
      perLapPlot[0].length = longest(perlapcut(graphs[i], delta(remover
```

```
    console.log(graphs[70]);
    g.push(new Dygraph(
      document.getElementById("graph" + i),
      vflip(rotate(perLapPlot)),
      {animatedZooms: true,
        //labels: [dataArr[0][0], dataArr[0][i]],
        ylabel: dataArr[0][i],
        //visibility: visibility_array,
        //axes: {x: {drawAxis: false, valueFormatter: function(col) {
        axes: {x: {drawAxis: false}},
        drawGrid: false,
        plugins: [new Dygraph.Plugins.Crosshair({direction: "vertical
        legend: 'follow'}
      ));
      //perLapPlot.length = 1
    }
  }
  for (var i = 0; i < removenear(laps, 50).length; i++){
    addRow(i,sec2time(graphs[0][0][delta(lapsArray)[i]]), sec2time(gr
    console.log(graphs[0][0]);
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
  });
  currentDiv = "container1lap";
});


container2lapbutton.addEventListener('click', event => {
  table.innerHTML = "";
  destroy(g);
  var lat = 52.4932793;
  var long = -1.1167698;
  var head = 90.02;
  var accuracy = 4;
  var laps = [0];
  for(var i = 0; i < cords[0].length; i++){
    if(cords[0][i].toFixed(accuracy) == lat.toFixed(accuracy)){
      if(cords[1][i].toFixed(accuracy) == long.toFixed(accuracy)){
```

```
      laps.push(i);
    }
  }
}
for (var i = 0; i < graphs.length; i++){
  if(dataArr[1][i] == "engine"){
    document.getElementById("container2lap").innerHTML+=('<div id='
  }
}
var lapsArray = removenear(laps, 50);
for (var i = 0; i < graphs.length; i++){
  if(dataArr[1][i] == "engine"){
    perLapPlot = perlapcut(graphs[i], delta(removenear(laps, 50)));
    perLapPlot.unshift(graphs[0][0].slice());
    console.log(graphs[70]);
    perLapPlot[0].length = longest(perlapcut(graphs[i], delta(remov
    console.log(graphs[70]);
    g.push(new Dygraph(
      document.getElementById("graph" + i),
      vflip(rotate(perLapPlot)),
      {animatedZooms: true,
        //labels: [dataArr[0][0], dataArr[0][i]],
        ylabel: dataArr[0][i],
        //visibility: visibility_array,
        //axes: {x: {drawAxis: false, valueFormatter: function(col
        axes: {x: {drawAxis: false}},
        drawGrid: false,
        plugins: [new Dygraph.Plugins.Crosshair({direction: "vertic
        legend: 'follow'}
      ));
      //perLapPlot.length = 1
    }
  }
  for (var i = 0; i < removenear(laps, 50).length; i++){
    addRow(i,sec2time(graphs[0][0][delta(lapsArray)[i]]), sec2time(
    console.log(graphs[0][0]);
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
```

```
  });
  currentDiv = "container2lap";
});


container3lapbutton.addEventListener('click', event => {
  table.innerHTML = "";
  destroy(g);
  var lat = 52.4932793;
  var long = -1.1167698;
  var head = 90.02;
  var accuracy = 4;
  var laps = [0];
  for(var i = 0; i < cords[0].length; i++){
    if(cords[0][i].toFixed(accuracy) == lat.toFixed(accuracy)){
      if(cords[1][i].toFixed(accuracy) == long.toFixed(accuracy)){
        laps.push(i);
      }
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "chassis"){
      document.getElementById("container3lap").innerHTML+=('<div id
    }
  }
  var lapsArray = removenear(laps, 50);
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "chassis"){
      perLapPlot = perlapcut(graphs[i], delta(removenear(laps, 50))
      perLapPlot.unshift(graphs[0][0].slice());
      console.log(graphs[70]);
      perLapPlot[0].length = longest(perlapcut(graphs[i], delta(rem
      console.log(graphs[70]);
      g.push(new Dygraph(
        document.getElementById("graph" + i),
        vflip(rotate(perLapPlot)),
        {animatedZooms: true,
          //labels: [dataArr[0][0], dataArr[0][i]],
          ylabel: dataArr[0][i],
          //visibility: visibility_array,
          //axes: {x: {drawAxis: false, valueFormatter: function(co
```

```
        axes: {x: {drawAxis: false}},
        drawGrid: false,
        plugins: [new Dygraph.Plugins.Crosshair({direction: "vert
        legend: 'follow'}
      ));
      //perLapPlot.length = 1
    }
  }
  for (var i = 0; i < removenear(laps, 50).length; i++){
    addRow(i,sec2time(graphs[0][0][delta(lapsArray)[i]]), sec2tim
    console.log(graphs[0][0]);
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
  });
  currentDiv = "container3lap";
});



container4lapbutton.addEventListener('click', event => {
  table.innerHTML = "";
  destroy(g);
  var lat = 52.4932793;
  var long = -1.1167698;
  var head = 90.02;
  var accuracy = 4;
  var laps = [0];
  for(var i = 0; i < cords[0].length; i++){
    if(cords[0][i].toFixed(accuracy) == lat.toFixed(accuracy)){
      if(cords[1][i].toFixed(accuracy) == long.toFixed(accuracy))
        laps.push(i);
      }
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "elec"){
      document.getElementById("container4lap").innerHTML+=('<div
    }
```

```
}
var lapsArray = removenear(laps, 50);
for (var i = 0; i < graphs.length; i++){
  if(dataArr[1][i] == "elec"){
    perLapPlot = perlapcut(graphs[i], delta(removenear(laps, 50
    perLapPlot.unshift(graphs[0][0].slice());
    console.log(graphs[70]);
    perLapPlot[0].length = longest(perlapcut(graphs[i], delta(r
    console.log(graphs[70]);
    g.push(new Dygraph(
      document.getElementById("graph" + i),
      vflip(rotate(perLapPlot)),
      {animatedZooms: true,
        //labels: [dataArr[0][0], dataArr[0][i]],
        ylabel: dataArr[0][i],
        //visibility: visibility_array,
        //axes: {x: {drawAxis: false, valueFormatter: function(
        axes: {x: {drawAxis: false}},
        drawGrid: false,
        plugins: [new Dygraph.Plugins.Crosshair({direction: "ve
        legend: 'follow'}
      ));
      //perLapPlot.length = 1
    }
  }
  for (var i = 0; i < removenear(laps, 50).length; i++){
    addRow(i,sec2time(graphs[0][0][delta(lapsArray)[i]]), sec2t
    console.log(graphs[0][0]);
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
  });
  currentDiv = "container4lap";
});



container5lapbutton.addEventListener('click', event => {
  table.innerHTML = "";
```

```
destroy(g);
var lat = 52.4932793;
var long = -1.1167698;
var head = 90.02;
var accuracy = 4;
var laps = [0];
for(var i = 0; i < cords[0].length; i++){
  if(cords[0][i].toFixed(accuracy) == lat.toFixed(accuracy)){
    if(cords[1][i].toFixed(accuracy) == long.toFixed(accuracy
      laps.push(i);
    }
  }
}
for (var i = 0; i < graphs.length; i++){
  if(dataArr[1][i] == "gforce"){
    document.getElementById("container5lap").innerHTML+=('<di
  }
}
var lapsArray = removenear(laps, 50);
for (var i = 0; i < graphs.length; i++){
  if(dataArr[1][i] == "gforce"){
    perLapPlot = perlapcut(graphs[i], delta(removenear(laps,
    perLapPlot.unshift(graphs[0][0].slice());
    console.log(graphs[70]);
    perLapPlot[0].length = longest(perlapcut(graphs[i], delta
    console.log(graphs[70]);
    g.push(new Dygraph(
      document.getElementById("graph" + i),
      vflip(rotate(perLapPlot)),
      {animatedZooms: true,
        //labels: [dataArr[0][0], dataArr[0][i]],
        ylabel: dataArr[0][i],
        //visibility: visibility_array,
        //axes: {x: {drawAxis: false, valueFormatter: functio
        axes: {x: {drawAxis: false}},
        drawGrid: false,
        plugins: [new Dygraph.Plugins.Crosshair({direction: '
        legend: 'follow'}
      ));
      //perLapPlot.length = 1
    }
```

```
  }
  for (var i = 0; i < removenear(laps, 50).length; i++){
    addRow(i,sec2time(graphs[0][0][delta(lapsArray)[i]]), sec
    console.log(graphs[0][0]);
  }
  Dygraph.synchronize(g.slice(1), {
    zoom: true,
    selection: true,
    range: false
  });
  currentDiv = "container5lap";
});



container6lapbutton.addEventListener('click', event => {
  table.innerHTML = "";
  destroy(g);
  var lat = 52.4932793;
  var long = -1.1167698;
  var head = 90.02;
  var accuracy = 4;
  var laps = [0];
  for(var i = 0; i < cords[0].length; i++){
    if(cords[0][i].toFixed(accuracy) == lat.toFixed(accuracy)
      if(cords[1][i].toFixed(accuracy) == long.toFixed(accura
        laps.push(i);
      }
    }
  }
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "gps"){
      document.getElementById("container6lap").innerHTML+=('<
    }
  }
  var lapsArray = removenear(laps, 50);
  for (var i = 0; i < graphs.length; i++){
    if(dataArr[1][i] == "gps"){
      perLapPlot = perlapcut(graphs[i], delta(removenear(laps
      perLapPlot.unshift(graphs[0][0].slice());
      console.log(graphs[70]);
```

```
      perLapPlot[0].length = longest(perlapcut(graphs[i], del
      console.log(graphs[70]);
      g.push(new Dygraph(
        document.getElementById("graph" + i),
        vflip(rotate(perLapPlot)),
        {animatedZooms: true,
          //labels: [dataArr[0][0], dataArr[0][i]],
          ylabel: dataArr[0][i],
          //visibility: visibility_array,
          //axes: {x: {drawAxis: false, valueFormatter: funct
          axes: {x: {drawAxis: false}},
          drawGrid: false,
          plugins: [new Dygraph.Plugins.Crosshair({direction:
          legend: 'follow'}
        ));
        //perLapPlot.length = 1
      }
    }
    for (var i = 0; i < removenear(laps, 50).length; i++){
      addRow(i,sec2time(graphs[0][0][delta(lapsArray)[i]]), s
      console.log(graphs[0][0]);
    }
    Dygraph.synchronize(g.slice(1), {
      zoom: true,
      selection: true,
      range: false
    });
    currentDiv = "container6lap";
  });
```

10.2 index.js

```
const {app, BrowserWindow, Menu} = require('electron');
const electron = require('electron');
const ipc = require('electron').ipcMain;
// Module to control application life.
// Module to create native browser window.

const path = require('path');
const url = require('url');

let mainWindow;
//let importWindow;

// This method will be called when Electron has done everything
// initialization and ready for creating browser windows.
app.on('ready', function() {
    // Create the browser window.
    mainWindow = new BrowserWindow({width: 1800, height: 1200});
    //importWindow = new BrowserWindow({width: 500, height: 400, frame: false, show: fals

    mainWindow.webContents.openDevTools();

    // and load the index.html of the app.
    mainWindow.loadURL('file://' + __dirname + '/index.html');
    //importWindow.loadURL('file://' + __dirname + '/importwindow.html');

    var menu = Menu.buildFromTemplate([
    {
        label: 'File',
            submenu: [
            {label:'Import',
            click() {
              openImportWindow();
            },
            accelerator: 'CmdOrCtrl+I'},
            {type: 'separator'},
            {label:'Exit',
            click() {app.quit()},
            accelerator: 'CmdOrCtrl+Q'}
```

```
            ]
        }
      ])
      Menu.setApplicationMenu(menu);

    // Emitted when the window is closed.
    mainWindow.on('closed', function() {
        // Dereference the window object, usually you would store windows
        // in an array if your app supports multi windows, this is the time
        // when you should delete the corresponding element.
        mainWindow = null;
        //importWindow = null;
    });
});

// Quit when all windows are closed.
app.on('window-all-closed', function() {
    if (process.platform != 'darwin')
        app.quit();
});

ipc.on('reply', (event, message) => {
        console.log(event, message);
        mainWindow.webContents.send('messageFromMain', `${message}`);
})

function openImportWindow(){
  let win = new BrowserWindow({width:800, height: 500, frame: false, parent: mainWindow ,
          win.webContents.on('did-finish-load', () => {
                //in.webContents.send('message', 'Well hello there my good sir')
        });
        win.webContents.openDevTools();
        win.on('close', () => {
          win = null;
        });
        win.loadURL(path.join('file://', process.cwd(), 'import_window.html'));
        win.show();
}
```

10.3 import_window.js

```
ipc = require('electron').ipcRenderer;
ipc.on('message', (event, message) => console.log(message));
process.env['ELECTRON_DISABLE_SECURITY_WARNINGS'] = 'true';
const fs = require('fs');
var inputFile = '';
var parse = require('csv-parse');
var column_choices = [];
const inputElement = document.getElementById("input");
const importButton = document.getElementById("import-button");
const csvDiv = document.getElementById("csv-rows-div");


inputElement.addEventListener("change", handleFiles, false);
function handleFiles() {
  const fileList = this.files;
  inputFile = fileList[0].path;
  var parser = parse({delimiter: ','}, function(err, data){
    for (i = 0; i < data[0].length; i++) {
      csvDiv.innerHTML+=('<div id="row' + i + '">' + data[0][i] + '<select id="import-sel
    }
  });
  fs.createReadStream(inputFile).pipe(parser);
}

importButton.addEventListener('click', event => { //this function creates an array with t
  for (i = 0; i < document.getElementsByTagName("select").length; i++) {
    var e = document.getElementById("import-selection" + i);
    column_choices.push(e.options[e.selectedIndex].value);
  }
  var output = {
    column: column_choices,
    data: inputFile
  };
  console.log(output);
  ipc.send('reply', JSON.stringify(output));
  window.close();
})
//ipc.on('inputFile', (event, message) => console.log(message));
```

10.4 index.html

```
<!DOCTYPE HTML>
<html>
<head>
  <script src="./dygraphs.js"></script>
  <script src="./extras/sync.js"></script>
  <script src="./extras/crosshair.js"></script>
  <script src="./extras/gauge.js"></script>
  <script src="./extras/interaction.js"></script>
  <link rel="stylesheet" type="text/css" href="style.css">
  <link rel="stylesheet" type="text/css" href="./extras/extrastyle.css">
</head>
<body style="background-color: #121212">
  <div class="w3-container">
    <div class="w3-bar w3-black">
      <button class="w3-bar-item w3-button" onclick="openTab('main')">Overview</button>
      <button class="w3-bar-item w3-button" id="perlaptab" onclick="openTab('perlap')">La
      <button class="w3-bar-item w3-button" id="extrastab" onclick="openTab('extras')">Qu
    </div>
    <div id="main" class="w3-container tabs">
      <div class="w3-bar w3-black">
        <button id="container1button" class="w3-bar-item w3-button">Driver Controls</butt
        <button id="container2button" class="w3-bar-item w3-button">Engine</button>
        <button id="container3button" class="w3-bar-item w3-button">Chassis</button>
        <button id="container4button" class="w3-bar-item w3-button">Elec</button>
        <button id="container5button" class="w3-bar-item w3-button">G-Forces</button>
        <button id="container6button" class="w3-bar-item w3-button">GPS</button>
        <button id="playbutton" class="w3-bar-item w3-button">Slow</button>
        <button id="speedbutton" class="w3-bar-item w3-button">Fast</button>
        <button id="pausebutton" class="w3-bar-item w3-button" disabled>Pause</button>
      </div>
      <div id="base_container" class="w3-row" >
        <div class="w3-container">
          <div id="container1"></div>
          <div id="container2"></div>
          <div id="container3"></div>
          <div id="container4"></div>
          <div id="container5"></div>
          <div id="container6"></div>
```

```
    </div>
  </div>
    <div id="gaugediv" class="w3-container w3-row">
      <div class="w3-threequarter">
        <div class="w3-container w3-quarter">
          <canvas id="throttlegauge" width="140%" style="display: block; margin: 0 au
          <span id="throttle-gauge-value" style="display: flex;justify-content: cente
        </div>
        <div class="w3-container w3-quarter">
          <canvas id="brakegauge" width="140%" style="display: block; margin: 0 auto;
          <span id="brake-gauge-value" style="display: flex;justify-content: center;
        </div>
        <div class="w3-container w3-quarter">
          <canvas id="speedgauge" width="140%" style="display: block; margin: 0 auto;
          <span id="speed-gauge-value" style="display: flex;justify-content: center;
        </div>
        <div class="w3-container w3-quarter">
          <canvas id="steeringgauge" width="200%" style="display: block; margin: 0 au
          <span id="steering-gauge-value" style="display: flex;justify-content: cente
        </div>
      </div>
      <div class="w3-container w3-quarter">
        <div id="map_container"></div>
      </div>
    </div>
</div>
<div id="perlap" class="w3-container tabs" style="display:none">
  <div class="w3-bar w3-black">
    <button id="container1lapbutton" class="w3-bar-item w3-button">Overview</button>
    <button id="container2lapbutton" class="w3-bar-item w3-button">Engine</button>
    <button id="container3lapbutton" class="w3-bar-item w3-button">Chassis</button>
    <button id="container4lapbutton" class="w3-bar-item w3-button">Elec</button>
    <button id="container5lapbutton" class="w3-bar-item w3-button">G-Forces</button>
    <button id="container6lapbutton" class="w3-bar-item w3-button">GPS</button>
  </div>
  <div id="perlap_container" class="w3-row" >
    <div class="w3-container">
      <div id="container1lap"></div>
      <div id="container2lap"></div>
      <div id="container3lap"></div>
      <div id="container4lap"></div>
```

```
          <div id="container5lap"></div>
          <div id="container6lap"></div>
        </div>
      </div>
        <div id="laptablediv" class="w3-container w3-quarter">
          <table id="table" class="w3-table-all">
            <thead>
              <th id="showlap"></th>
              <th>Lap</th>
              <th>Total Time</th>
              <th>Lap Time</th>
            </thead>
          </table>
        </div>
    </div>
    <div id="extras" class="w3-container tabs" style="display:none">
      <div class="w3-bar">
        <select id="selectX" class="w3-bar-item">
        </select>
        <select id="selectY" class="w3-bar-item">
        </select>
        <button id="quickplotbutton" class="w3-bar-item w3-button">Quick Plot</button>
      </div>
      <div id="quickplot_container">
      </div>
    </div>
  </div>
  <script>
    function openTab(tab) {
      var i;
      var x = document.getElementsByClassName("tabs");
      for (i = 0; i < x.length; i++) {
        x[i].style.display = "none";
      }
      document.getElementById(tab).style.display = "block";
    }
  </script>
  <script src="./app.js"></script>
</body>
</html>
```

10.5 import_window.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Hello World!</title>
  </head>
  <body style="background-color: lightblue;">
    <div style="margin: 5em;">
    <input type="file" id="input" accept=".csv" multiple>
    <button id="import-button">Import</button>
    </div>
    <div id="csv-rows-div">
    </div>
  </body>
  <script>
    require('./import_window.js');
  </script>
</html>
```

10.6 style.css

```
/**
 * Default styles for the dygraphs charting library.
 */

.dygraph-legend {
  position: absolute;
  font-size: 19px;
  z-index: 1;
  background: lightblue;
  border: 1px solid black;
  /* opacity: 0.75; */
  line-height: normal;
  overflow: hidden;
}

/* styles for a solid line in the legend */
.dygraph-legend-line {
  display: inline-block;
  position: relative;
  bottom: .5ex;
  padding-left: 1em;
  height: 1px;
  border-bottom-width: 2px;
  border-bottom-style: solid;
  /* border-bottom-color is set based on the series color */
}

/* styles for a dashed line in the legend, e.g. when strokePattern is set */
.dygraph-legend-dash {
  display: inline-block;
  position: relative;
  bottom: .5ex;
  height: 1px;
  border-bottom-width: 2px;
  border-bottom-style: solid;
  /* border-bottom-color is set based on the series color */
  /* margin-right is set based on the stroke pattern */
  /* padding-left is set based on the stroke pattern */
```

```
}

.dygraph-roller {
  position: absolute;
  z-index: 10;
}

/* This class is shared by all annotations, including those with icons */
.dygraph-annotation {
  position: absolute;
  z-index: 10;
  overflow: hidden;
}

/* This class only applies to annotations without icons */
/* Old class name: .dygraphDefaultAnnotation */
.dygraph-default-annotation {
  border: 1px solid black;
  background-color: white;
  text-align: center;
}

.dygraph-axis-label {
  /* position: absolute; */
  /* font-size: 14px; */
  z-index: 10;
  line-height: normal;
  overflow: hidden;
  color: white;  /* replaces old axisLabelColor option */
}

.dygraph-axis-label-x {
  color: white;  /* replaces old axisLabelColor option */
}

.dygraph-axis-label-y {
  color: white;  /* replaces old axisLabelColor option */

}

.dygraph-axis-label-y2 {
```

```
}

.dygraph-title {
  font-weight: bold;
  z-index: 10;
  text-align: center;
  /* font-size: based on titleHeight option */
}

.dygraph-xlabel {
  text-align: center;
  font-color: white;
  /* font-size: based on xLabelHeight option */
}

/* For y-axis label */
.dygraph-label-rotate-left {
  text-align: center;
  /* See http://caniuse.com/#feat=transforms2d */
  transform: rotate(90deg);
  -webkit-transform: rotate(90deg);
  -moz-transform: rotate(90deg);
  -o-transform: rotate(90deg);
  -ms-transform: rotate(90deg);
}

/* For y2-axis label */
.dygraph-label-rotate-right {
  text-align: center;
  color: white;
  /* See http://caniuse.com/#feat=transforms2d */
  transform: rotate(-90deg);
  -webkit-transform: rotate(-90deg);
  -moz-transform: rotate(-90deg);
  -o-transform: rotate(-90deg);
  -ms-transform: rotate(-90deg);
}
```

10.7 package.json

```
{
  "name": "electron-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "2d-array-rotation": "^2.1.1",
    "chart.js": "^2.9.3",
    "csv-parse": "^4.8.8",
    "downsample-lttb": "0.0.1",
    "dygraphs": "^2.1.0",
    "electron": "^2.0.3",
    "graph.js": "^1.21.1"
  },
  "devDependencies": {},
  "scripts": {
    "start": "electron ."
  },
  "author": "",
  "license": "ISC"
}
```

Bibliography

[1] IMechE. 2019. 2020 Rules. [PDF FILE] Available at: https://www.imeche.org/docs/default-source/1-oscar/formula-student/fs2020/rules/formulastudentrules2020.pdf?sfvrsn=2. [Accessed 17 March 2020].

[2] Sull, D. and Sull, C. 2018 With Goals, FAST Beats SMART. [PDF FILE] Available at: http://shorelinelabs.org/wp-content/uploads/2019/10/with-goals-fast-beats-smart.pdf [Accessed 1 March 2020].

[3] Murphy, N. 2003 Serial Communication Protocols: CAN vs. SPI. [ONLINE] Available at: https://barrgroup.com/embedded-systems/how-to/can-vs-spi [Accessed 4 April 2020].

[4] AVR Freaks. 2017. CAN vs I2C vs SPI. [ONLINE] Available at: https://www.avrfreaks.net/forum/can-vs-i2c-vs-spi. [Accessed 18 May 2020].

[5] AVR Freaks. 2009. SPI vs I2C Protocol. [ONLINE] Available at: https://bitwizard.nl/wiki/SPI_versus_I2C_protocols. [Accessed 2 February 2020].

[6] AVR Freaks. 2009. SPI vs I2C Protocol. [ONLINE] Available at: https://bitwizard.nl/wiki/SPI_versus_I2C_protocols. [Accessed 2 February 2020].

[7] Practicalee. 2014. SPI. [ONLINE] Available at: https://practicalee.com/spi/. [Accessed 31 March 2020].

[8] CAN-CiA. 2011. History of CAN Technology. [ONLINE] Available at: https://www.can-cia.org/can-knowledge/can/can-history/. [Accessed 9 April 2020].

[9] Search Security. 2005. Checksum. [ONLINE] Available at: https://searchsecurity.techtarget.com/definition/checksum. [Accessed 5 April 2020].

[10] Geeks For Geeks. 2015. Back-Off Algorithm for CSMA. [ONLINE] Available at: https://www.geeksforgeeks.org/back-off-algorithm-csmacd/. [Accessed 1 February 2020].

[11] Corrigan, S., 2002. Introduction to the Controller Area Network (CAN). 2nd ed.: Texas Instruments.

[12] Elektromotus. 2006. CAN bus Quick Start Guide v0.2 rc3. [PDF FILE] Available at: https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwjYi9vNj6Tp SBUIHQ-WCVgQFjAAegQIARAB&url=http%3A%2F%2Fsd94f85fb6682eee7.jimcontent.com%2Fdownl [Accessed 26 March 2020].

[13] Comparitech. 2019. Network Topology: 6 Network Topologies Explained & Compared. [ONLINE] Available at: https://www.comparitech.com/net-admin/network-topologies-advantages-disadvantages//#Advantages-3. [Accessed 12 March 2020].

[14] Gigamon. 2013. Understanding Single Points of Failure. [ONLINE] Available at: https://blog.gigamon.com/2018/08/31/understanding-single-points-of-failure/. [Accessed 1 May 2020].

[15] Axiomatic. 2006. Q&A - What is CAN. [PDF FILE] Available at: https://www.axiomatic.com/whatiscan.pdf. [Accessed 3 April 2020].

[16] On Set Comp. 2011. Data Logger Basics. [PDF FILE] Available at: https://www.onsetcomp.com/files/data-logger-basics.pdf. [Accessed 24 March 2020].

[17] GeoScientific. 2008. Data Logger Fundamentals for Environmental Monitoring Applications. [PDF FILE] Available at: https://pdfs.semanticscholar.org/4746/5c56c8c088b6c291ce6ebf14ddf09238b930.pdf. [Accessed 6 May 2020].

[18] Lopez, G. and Seaber, A., 2009. THE THEORY AND PRACTICE OF RACE-VEHICLE DATAACQUISITION AND ANALYSIS IN MOTOR-SPORTS ENGINEERING EDUCATION. 1st ed. Old Dominion University: American Society for Engineering Education.

[19] EL PRO CUS. 2018. Types of Analog and Digital Sensors with Applications. [ONLINE] Available at: https://www.elprocus.com/types-analog-digital-sensors/. [Accessed 4 May 2020].

[20] D. Thomas, K., M. Bockhorst, R., L. Mauck, J. and L. Quinn, E., 2016. Digital Sensor Technology. 9th ed. Idaho National Laboratory: Idaho National Laboratory.

[21] Information In. 2013. The Meaning Behind the SONY VAIO Logo. [ONLINE] Available at: https://www.informationin.com/2013/04/the-meaning-behind-sony-vaio-logo.html. [Accessed 20 January 2020].

[22] Arrow. 2013. Basics of Analog-to-Digital Converters. [ONLINE] Available at: https://www.arrow.com/en/research-and-events/articles/engineering-resource-basics-of-analog-to-digital-converters. [Accessed 4 March 2020].

[23] Baker E. Open source data logger for low-cost environmental monitoring. Biodivers Data J. 2014;(2):e1059. Published 2014 Feb 11. doi:10.3897/BDJ.2.e1059

[24] Atlas RFID Store. 2017. What is Telemetry. [ONLINE] Available at: https://www.atlasrfidstore.com/rfid-insider/what-is-telemetry. [Accessed 25 March 2020].

[25] Bugeja, Keith & Spina, Sandro & Buhagiar, François. 2017. Telemetry-based Optimisation for User Training in Racing Simulators. 10.1109/VS-GAMES.2017.8055808.

[26] How Stuff Works. 2015. How Stock Car Telemetry Works. [ONLINE] Available at: https://auto.howstuffworks.com/auto-racing/nascar/nascar-basics/stock-car-telemetry.htm. [Accessed 3 May 2020].

[27] F1i. 2018. F1 Telemetry Data. [ONLINE] Available at: https://f1i.com/magazine/73067-f1-telemetry-data-race.html/3. [Accessed 9 March 2020].

[28] Racing Car Dynamics. 2014. DATA ACQUISITION: 6 REASONS TO USE AND THE FUNDAMENTALS YOU MUST KNOW. [ONLINE] Available at: https://racingcardynamics.com/data-acquisition-fundamentals/. [Accessed 2 March 2020].

[29] IO Technologies. 2018. Data Visualization Trends. [ONLINE] Available at: https://iotechnologies.com/blog/data-visualization-trends. [Accessed 11 April 2020].

[30] Institude of Physics. 2016. The Rise and Fall of Data Science. [ONLINE] Available at: https://beta.iop.org/rise-and-rise-data-science. [Accessed 14 April 2020].

[31] Segers, J., 2014. Analysis Techniques for Racecar Data Acquisition. 2nd ed. Pennsylvania: SAE International.

[32] Braune, N., 2014. Telemetry Unit for a Formula StudentRace Car. 1st ed. Institut für Technische Informatik und Kommunikationsnetze: Swiss Federal Institute of Technology, Zurich.

[33] Wikimedia. 2013. Waterfall Model. [ONLINE] Available at: https://upload.wikimedia.org/wikipedia/commons/thumb/e/e2/Waterfall_model.svg/1024px-Waterfall_model.svg.png. [Accessed 3 January 2020].

[34] Acodez. 2018. 12 Best Software Development Methodologies with Pros and Cons. [ONLINE] Available at: https://acodez.in/12-best-software-development-methodologies-pros-cons/#Waterfall_Model. [Accessed 3 January 2020].

[35] MathWorks. 2006. MathWorks. [ONLINE] Available at: https://uk.mathworks.com/?s_tid=gn_logo. [Accessed 26 February 2020].

[36] MathWorks. 2016. App Designer's editor is slow and gets stuck alot. [ONLINE] Available at: https://uk.mathworks.com/matlabcentral/answers/279042-app-designer-s-editor-is-slow-and-gets-stuck-alot. [Accessed 10 February 2020].

[37] RStudio. 2015. Rstudio. [ONLINE] Available at: https://rstudio.com/. [Accessed 21 February 2020].

[38] RStudio. 2020. How does Shiny work. [ONLINE] Available at: https://support.rstudio.com/hc/en-us/articles/218294767-How-does-Shiny-work-. [Accessed 2 February 2020].

[39] Google Trends. 2006. Google Trends. [ONLINE] Available at: https://trends.google.com. [Accessed 13 March 2020].

[40] Hacker Noon. 2017. Electron: The Bad Parts. [ONLINE] Available at: https://hackernoon.com/electron-the-bad-parts-2b710c491547. [Accessed 9 April 2020].

[41] EDUCBA. 2018. SVG vs Canvas. [ONLINE] Available at: https://www.educba.com/svg-vs-canvas/. [Accessed 10 March 2020].

[42] EDUCBA. 2020. Comparing HTML5 Canvas VS SVG for Charting. [ONLINE] Available at: https://www.barchart.com/solutions/company/blog/3852318/comparing-html5-canvas-vs-svg-for-charting. [Accessed 1 May 2020]

[43] Steinarsson, S., 2013. Downsampling Time Series forVisual Representation. 1st ed. Faculty of Industrial Engineering, Mechanical Engineering and Computer Science: University of Iceland.

[44] UX Planet. 2019. 8 Tips for Dark Theme Design. [ONLINE] Available at: https://uxplanet.org/8-tips-for-dark-theme-design-8dfc2f8f7ab6. [Accessed 27 April 2020].

[45] Software Testing Fundamentals. 2014. System Testing. [ONLINE] Available at: http://softwaretestingfundamentals.com/system-testing/. [Accessed 16 April 2020].

[46] Software Testing Fundamentals. 2014. Black Box Testing. [ONLINE] Available at: http://softwaretestingfundamentals.com/black-box-testing/. [Accessed 16 April 2020].

[47] Try QA. 2013. What is Prototype model- advantages, disadvantages and when to use it?. [ONLINE] Available at: http://tryqa.com/what-is-prototype-model-advantages-disadvantages-and-when-to-use-it/. [Accessed 1 April 2020].

[48] Wikimedia. 2011. Three Software Development Patterns Mached Together. [ONLINE] Available at: https://upload.wikimedia.org/wikipedia/commons/thumb/5/5f/Three_software_developr Three_software_development_patterns_mashed_together.svg.png. [Accessed 3 February 2020].

[49] GNU. 2007. GNU General Public License. [ONLINE] Available at: https://www.gnu.org/licenses/gpl-3.0.en.html. [Accessed 28 February 2020].

[50] Fritzing. 2017. Guinea pig surveillance system. [ONLINE] Available at: http://fritzing.org/projects/guinea-pig-monitor. [Accessed 12 January 2020].

[51] Henry's Bench. 2018. Arduino CAN Bus Module 1st Network Tutorial. [ONLINE] Available at: http://henrysbench.capnfatz.com/henrys-bench/arduino-projects-tips-and-more/arduino-can-bus-module-1st-network-tutorial/. [Accessed 5 March 2020].

[52] Web Laboratorium. 2013. Temperature Measurement with aThermistor and an Arduino. [ONLINE] Available at: http://weblaboratorium.hu/wp-content/uploads/2016/05/thermistorArduino.pdf. [Accessed 18 May 2020].

[53] Arduino. 2019. SD Card Module with Arduino: How to Read/Write Data © GPL3+. [ONLINE] Available at: https://create.arduino.cc/projecthub/electropeak/sd-card-module-with-arduino-how-to-read-write-data-37f390. [Accessed 26 March 2020].

[54] Arduino. 2015. Memory. [ONLINE] Available at: https://www.arduino.cc/en/tutorial/memory. [Accessed 27 March 2020].