Lafayette College | Electrical and Computer Engineering

Critical Design Review Report

ECE 492 Spring 2017

Assembled by LFEV Team Spring 2017

Abstract

This document provides all materials or access to all materials relevant to the Critical Design Review for the Lafayette Formula Electric Vehicle for Spring 2017. This document begins with a system overview then describes all subsystems that have undergone major design iterations in this semester. These systems include: VSCADA, the Cell Application, TSI, GLV, and the Cooling System. Finally, a cost analysis and schedule are included for the overall LFEV team.

Table of Contents

System Design

System Overview

The objective for this system is to safely provide power to the motor in the electric vehicle. This is achieved by using 6 major subsystems. The first subsystem, tractive system voltage (TSV) has already been designed and it is just being implemented. The remaining 5 are in this document. The brains of the car is controlled by the Vehicle Supervisory Control and Data Acquisition (VSCADA) subsystem and it listens to the CAN network, the communication network between all subsystems in the car. This data can be accessed by a remote computer or by a cell phone app. This application shows information about the car as it operates. The tractive system interface (TSI) connects the high voltage to the motor controller safely. The cooling system allows the motor controller to operate at a safe temperature.

System States

The car has 5 major states. It is critical that it is clear how the car moves between the states.

The first state is off. While the car is in this state all electronics are turned off with the exception of the AMS and the PacMAN. These boards take power straight from the cells and do not have an off switch.

How to enter:

- GLVMS to off
- BRBLS pushed
- BRBRS pushed
- GLV battery completely discharged

The only state that the car can enter after a full shutdown is GLV powered. In this mode low voltage power is provided to all subsystems. The safety loop is armed and the AIRs could be armed in this state.

How to enter (AIRs off):

- BRBLS reset
- BRBRS reset
- MReset button on RS of car
- GLVMS on

At this point subsystems will be powered. Before drive mode can be entered the AIRs must be closed.

How to close the AIRs:

- All subsystems must close the safety loop
- BRB Cockpit reset
- CPR reset

- Driver Reset
- TSVMS on

With the AIRs closed drive mode can be entered.
To enter drive mode:

- Throttle must not be pressed
- Drive button pressed

Once in drive mode there are a few options. The first choice is to exit drive mode and return to the AIRs closed state.
To exit drive mode:

- Push the drive button

The throttle will disengage at this point and the car will come to a halt.

Another option is cruise control. This will hand control of the throttle over from the driver to VSCADA. TSI goes into more detail about how this works.
To enter cruise control:

- Push the cruise button

To exit cruise control:

- Push the cruise button

OR

- Tap the brake

It will be clear that cruise control has been left due to a light on the cockpit.

There are three failure modes. The first mode is a standard fault. Depending on where it happened in the system the driver may be able to reset it or they may need to get out of the car and push the reset button. The second mode is a brake overtravel. In this case not only do all of the reset buttons have to be pushed but also the brake overtravel needs to be reset. The final fault mode is loss of GLV power. From here a full reboot sequence needs to start from GLV off.

Overall Schematic

http://sites.lafayette.edu/ece492-sp17/overall-system-diagram/

Subsystem Designs

Vehicle Supervisory Control and Data Acquisition (VSCADA)

Overview

The SCADA software will be listening to a CANBus channel for packets sent from subsystems. The ID of the packet received will be used as the ID for storing the data received in the database. Both raw and calibrated data will be stored in the database to allow for easier comparisons to be made when re-calibrating and comparing data logs. Once the data is stored, all there is left to do is extract the data and send it out to be analyzed. An HTTP web server interacts with the database to pull information requested through the API. For a full explanation of the system, refer to the Software Maintainability Plan, in which you will find full documentation as well as the course of action being taken to ensure longevity.

Installation

The target operating system is Linux, specifically Raspbian and Ubuntu. A link on the website will download the latest snapshot. Anyone will have the capability to either download a compressed file containing our code or will be able to clone the code down from our GIT repo. Upon download/uncompressing the files, a user will be able to 'cd' into the VSCADA directory and use the provided makefile to install (make install) any necessary software. They may then either run ./scada (run the current SCADA software) or ./configuration (run the current ConfigurationUI software) and the software will automatically start. Once properly installed, the system shall automatically run necessary code on startup so there is no added effort on the part of the user.

CANBus Interface

The CAN interface has been changed from previous years. The old implementation, as seen in Figure 1 works however it is a little limited. As seen in the tables within Figure 1, each byte has a specific mapping for it's data value. This not only makes it more difficult from an expandability standpoint, it also requires a very specific format for every packet of information sent that both ends of the CAN communication must agree upon. In the new implementation, the software will continuously be listening on the CANBus interface as with the old system; however, it will be looking for key/value pairs. Every sensor in the system will have a unique ID, as mentioned in the overview and again in more detail in the Database section. The unique IDs will allow us to not only parameterize every sensor individually, it will allow us control to log sensors at different rates, send data in an arbitrary order and even mix who's sending information easily. With this method it doesn't particularly matter who is sending over CAN or what order they send the information as the key/value pairs, ID/data in our case, as they will be mapped

based on their ID and not the order they are received in. This protocol allows for rapid expansion as a user must only go into the configuration database and enter in the information of a new sensor and send the sensor ID as well as its data over the CAN line and then the system shall now be configured with the newly added sensor.

Generic CAN Messages from Controller

	ADDRESS ID						
	CAN ADDRESS 0x601	**Units**	**Scale**		**CAN ADDRESS 0x602**	**Units**	**Scale**
Byte0	Motor RPM high byte	RPM	1		Stator Frequency high byte	Hz	1
Byte1	Motor RPM low byte				Stator Frequency low byte		
Byte2	Motor Temp	Deg C	-40 to 200		Controller Fault Primary		
Byte3	Controller Temp				Controller Fault Secondary		
Byte4	RMS Current high byte	Amps	0.1		Throttle Input	%	1
Byte5	RMS Current low byte				Brake Input		
Byte6	Capacitor Voltage high byte	Volts	0.1		System Bits*		
Byte7	Capacitor Voltage low byte				Not used		

* System bits configuration	
Bit	**Logic**
0	Econo bit
1	Regen bit
2	Reverse bit
3	Brake Light Bit

Figure 1. Old Protocol for Transmission Over CAN - Dyno Example

Database

The database will consist of multiple data tables that will store various components about the SCADA system as a whole. This will include a primary table containing unique IDs for every sensor. The unique ID will serve as the primary key in our database and each table will have that primary key mapped to it for rapid access. This will limit the overall size of each table thus making queries faster as a whole and making data insertion easier. With this method, we will have normalized tables that can easily be expanded and rely more on inserting data and less on editing a specific entry. Data shows on the server instantaneously, once new data is pushed over the CAN, while the database will only update at a maximum rate of 60 times per second. We are confident that this is the proper way to handle things as with this structure we can have one reference/configuration table with many columns regarding the desired performance of a sensor, and update that only when new parameters are specified. Tables such as the data table will simply have the ID of the sensor we are logging, the raw data value, and the calibrated data per the user specified calibration parameters. A similar protocol will be implemented for all other tables such as the Error table which will track errors in the system. Each data point entered into the system will have a timestamp attached to it automatically. When searching for data, users

will be able to search based on a given subsystem, or subsystems, as well as a given date, or date range. The results returned will vary depending on the view that the user is currently in. For example, if the system is in drive mode, only select information will be accessed/viewable whereas in maintenance mode, everything is presented to the user.

Web Server

The HTTP web server is in charge of interacting with the database. To query the database, send a GET request to one of the endpoints provided in the API. There are four filter parameters you can set in a standard request to get only the information you want. These parameters are system type, ID number, start date, and end date. Alternatively, you can request only the most recent set of data. This will query the database for the most recent data entry from each individual subsystem. The server will then query the database with the parameters sent in the request and the database will respond. The server will convert the information into JSON format and send the packet to the client. The API will provide the functionality to query based on system, a specific date, a range of dates, or any combination of above. External applications such as the Cell App and the various operating modes will be able to interact with the web API to retrieve this information. For a full list of operations and how to use them, refer to the API.

Different Views

The different views available by the SCADA system shall directly align with the mode layouts set forth in the SOW. In Maintenance mode, all sensor values will be available to the user, both in calibrated and uncalibrated form. In this mode, all control functions, such as throttle or torque with the Dyno shall have the functionality to be altered. The ConfigurationUI will allow a user, with the password, the opportunity to change reference labels such as a sensors units, subsystem, calibration factors etc. This will be done through a series of input boxes and confirmation pages to ensure intended settings changes. This user shall also have the ability to enable or disable various safety checks and interlocks that constrain functionality and state transitions.

In Drive mode only the most essential data shall be displayed. The most essential data is effectively the data that is most important to the driver during a race. The current data represented on the screen is the data as set forth by the SOW. Drive mode shall be most useful for the dynamometer and vehicle/racing. The drive mode shall extend to a drive demo mode. It is in this mode that the system shall use the parameters determined by a user to generate/simulate real race data. Along with the system parameters, an onboard physics simulator shall be used to better simulate various torque and load conditions as though the car were actually being used to race.

The SCADA system shall also provide a Charging view which shall show key information of the packs and their SOC. The system shall in addition provide means for a long term shutdown. This will send codes to hardware peripherals signaling shutdown as well as close the database and terminate the web socket connection.

Cell Application

Overview

The cell phone application, called “LFEV-SCADA-Mobile” functions as an observation tool that is available through both android tablet and phone environments. The users of LFEV-SCADA-Mobile are people who are not necessarily knowledgeable about the LFEV system itself. The users will also be provided a tutorial and so we aim to build a software that can be used by basically anyone.

Key aspects of the project to be implemented:

- Application will be able to run both on tablets and cell phones
- Application Needs to use Java as the main language for easy accessibility for future teams
- Application uses JSON formatted data from a web-server for information gathering
- Application generates various types of displays for expanded functionality
- Application is able to display data as per the requirements in the Statement of Work R002k
- Application must abide by the Formula EV rules

Current priorities:

- Preset Charts/Diagrams -- High Priority
- Number Display -- High Priority
- Speedometer/Odometer -- High Priority
- Notifications + Threshold -- High Priority
- Log Messages -- High Priority
- Custom Chart Generation -- High Priority
- Time adjustment to Charts -- High Priority
- Dynamic View generation -- High Priority
- Tutorial -- High Priority
- Adding a new sensor that has same fields -- High Priority
- Adding an entirely new sensor that requires another field -- Medium Priority
- Creating preset displays -- Medium Priority
- Google Maps Coordinates -- Low Priority

Process View

In this section, some activity diagrams for the application are shown.

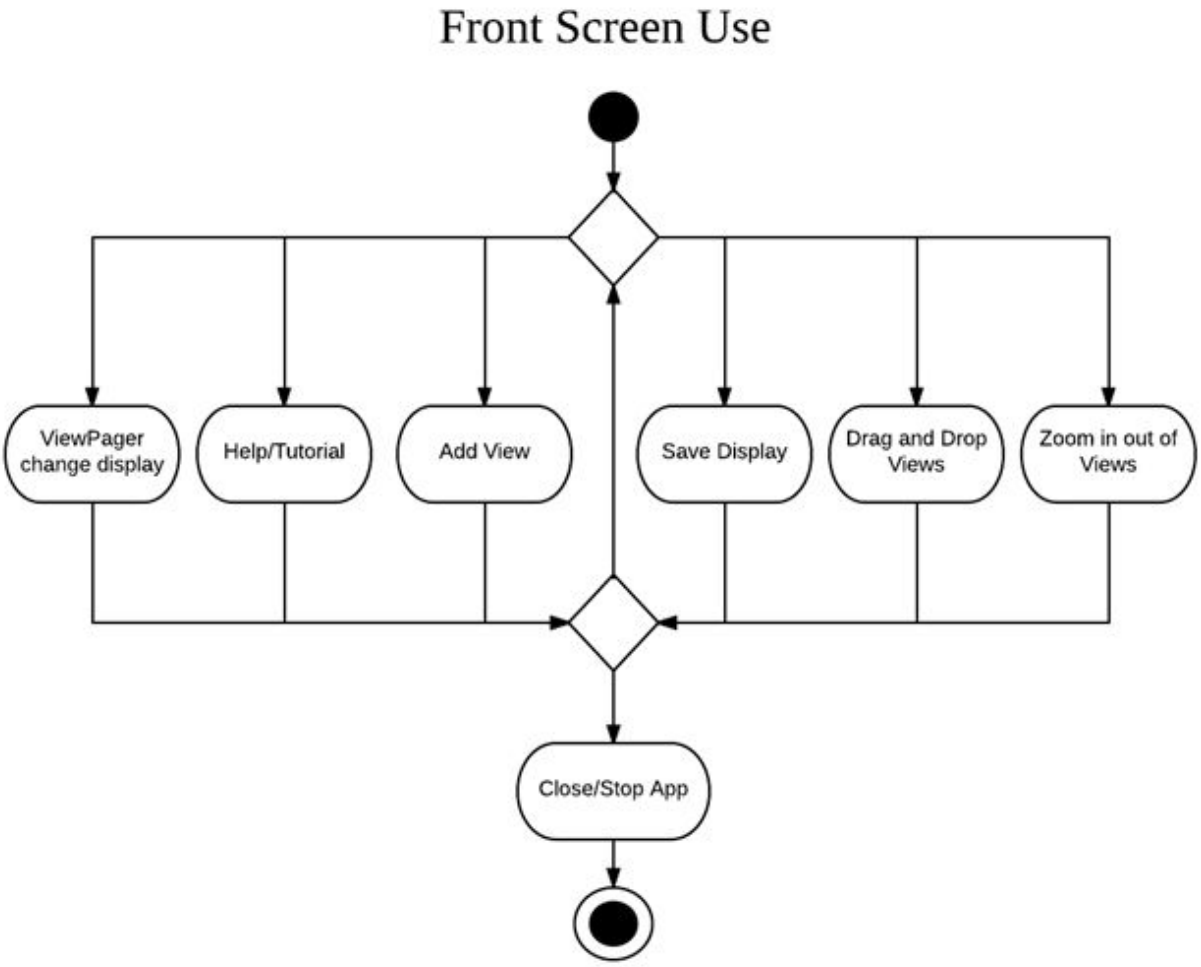


Figure 2. Front-Page Activity Diagram -- Shows what options the user will have access to and how the general application will update the display according to the choices of the user.

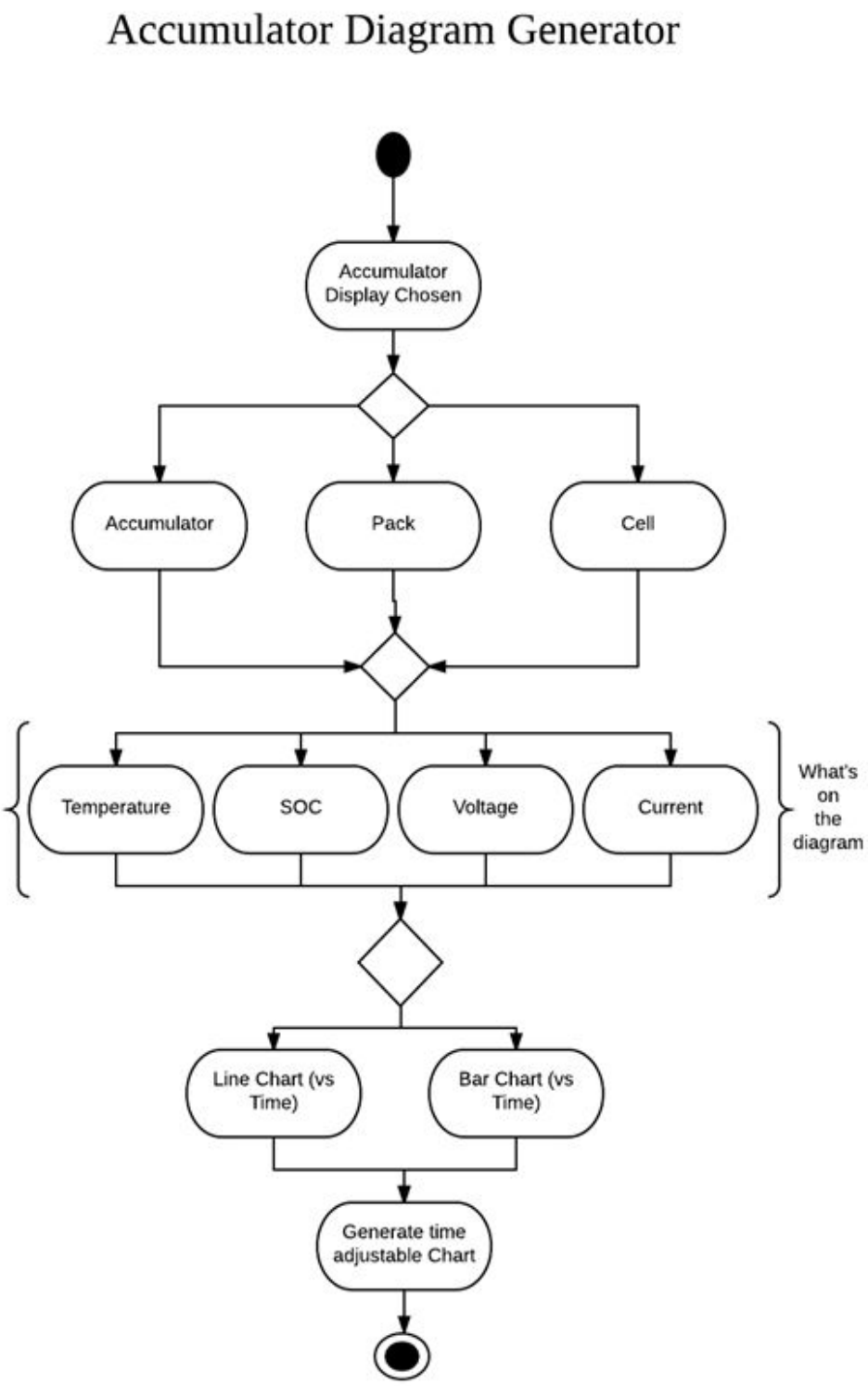


Figure 3. Accumulator Chart Generator Activity Diagram -- This shows how a chart will be generated according to the choices of the user. Examples of these are shown later in the document.

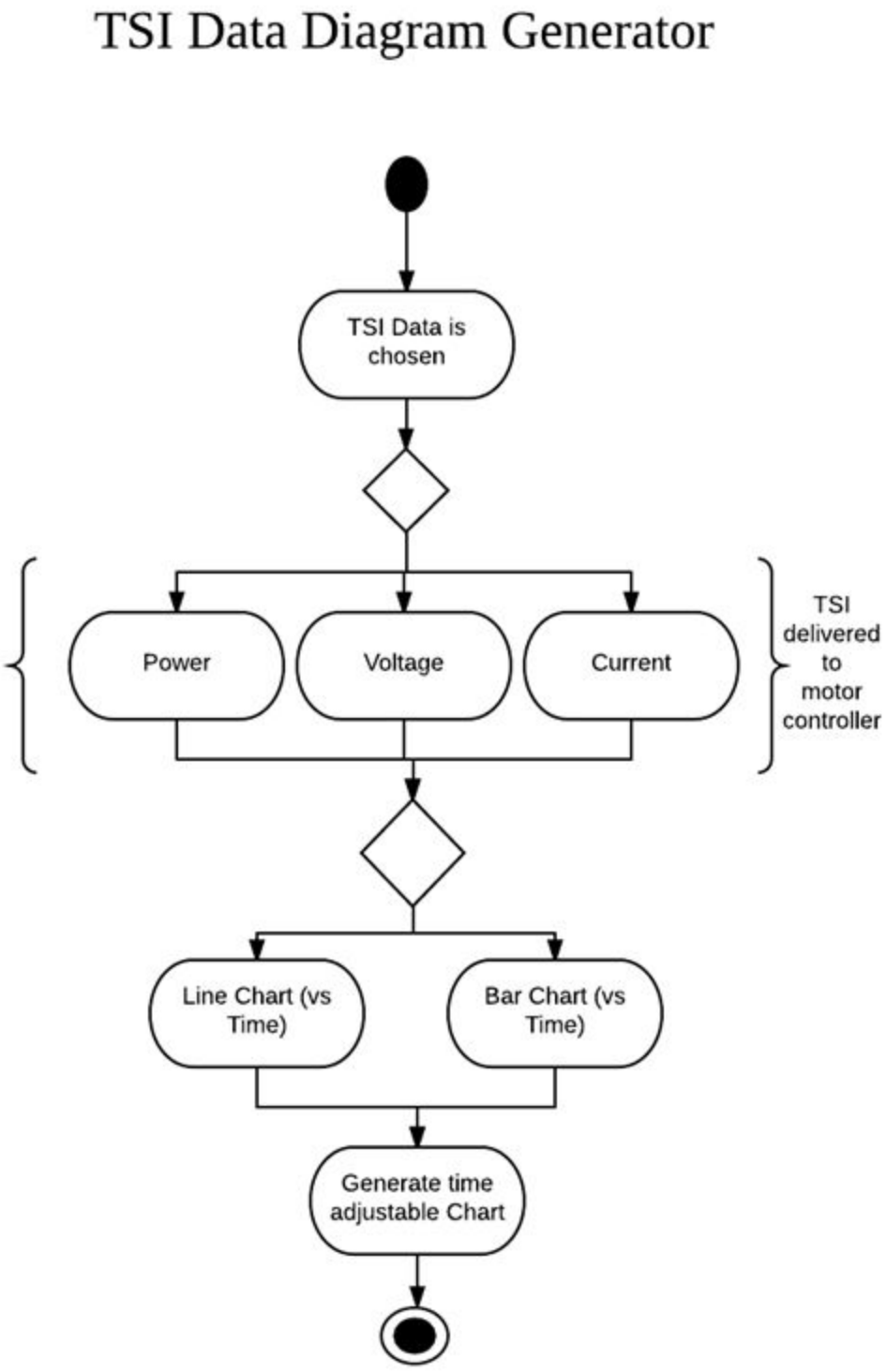


Figure 4. TSI Data Chart Generation Activity Diagram -- Similar to figure 2.3, shows the process for TSI Data charts

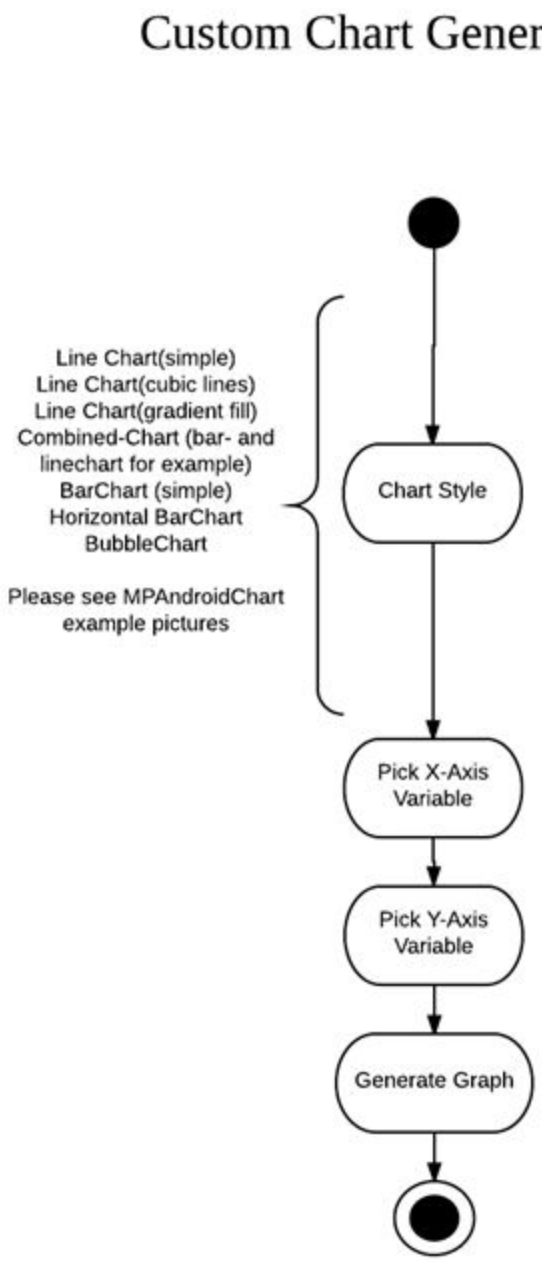


Figure .: Custom Chart Generation Activity Diagram -- Custom chart generation will be fairly simple but the options for the custom generation are not yet determined. Further documentation will be provided with tutorial and later during development.

Development View

Another important aspect of the application is the classes that will be created for the program. Here we have created a Content UML diagram:

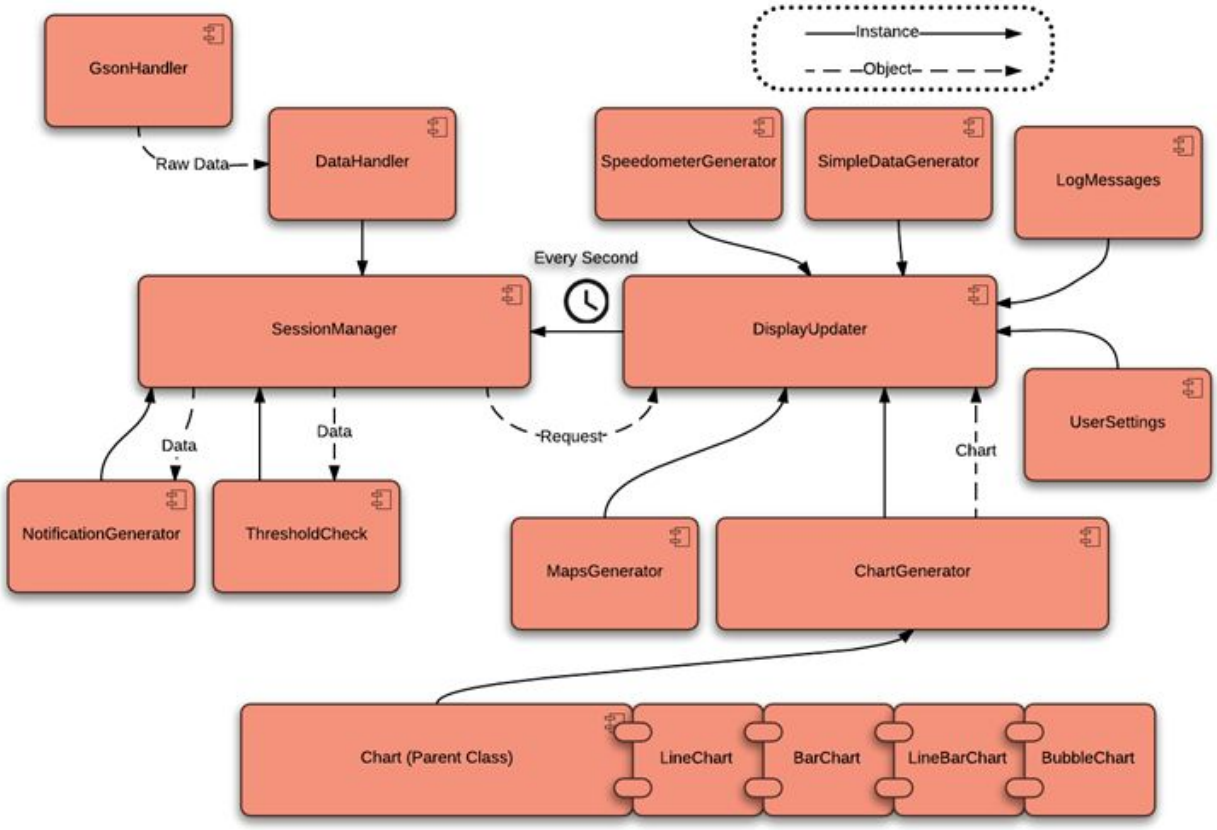


Figure 6. Development View -- Here you can see the classes and how they interact with each other in a very high level case. Each of these classes will be tested individually and the functionality will be demonstrated as the projects continues on. Two biggest components are DisplayUpdater and SessionManager.

Reference the Maintainability document for why we have decided on such a structure and how this provides expandability options in the application.

Data Structures

Here is an example of how the data will be provided to to the application from the SCADA team through the web-server:

```
{"subName":"TSV",
      "subSensors":{
      "1":{"sName":"Cell","sVal":"1.1","sUnits":"V"},
      "2":{"sName":"Cell","sVal":"3.7","sUnits":"V"},
      "3":{"sName":"Cell","sVal":"8.1","sUnits":"V"},
      "4":{"sName":"CellV4","sVal":"4.1","sUnits":"V"}
      }
}
```

Figure 7. Data from SCADA to Cell App

As you can see, the information is provided under fields. While the application will have a display that will make it easier to choose which data to display, the data structures will be simpler. We simply need a dictionary of values for each data for the provided information. HashMap is the most efficient way to implement dictionary for Java. For different levels of information we will need different dictionaries:

- One dictionary for the different timestamps provided by the SCADA system.
- Another dictionary for locating the specific sensor and the unit

Therefore, we will have a HashMap of HashMaps as our data structure in our application.

First, a higher level HashMap will have the combined name of the sensor as the “key” and the “value” will be the second HashMap. For example, “tsvcell1voltage” which is a lower case combination of System + Sensor + ID + Unit. This string will be used to create a hashcode that will determine where the measurements will be stored.

This HashMap will have the values as other HashMaps, which will have timestamps are the “key” and the Unit value as the “value”. For example, coming from “tsvcell1voltage”, “03/03/2017 17:56:34” will correspond to value of “8.1”. Let’s see it in an example code:

```
      HashMap<String, Hashmap<String, String>> allSystems = new HashMap<String,
HashMap<String, String>>;
      HashMap<String, String> newSystem = new HashMap<String, String>;
      newSystem.put(“03/03/2017 17:56:34”, “8.1”);
      allSystems.put(“tsvcell1voltage”, newSystem);
```

```
System.out.println(allSystems.get("tsvcell1voltage").get("03/03/2017
17:56:34"));
```

This block of code will store the voltage value for Cell 1 (which will be an internal designation) at the specific time it was recorded. Then it will print out the Unit value that we have specified.

Layout Details

Figure 8. This picture is taken with the apk run in a Moto G Osprey with the Android system of 7.1.

The charts and graphs generated by the application will have very simple functionality of being freely moveable.

In addition, these views are dynamically added to the display, meaning that the user will be able to add in any view that they would like to see and be able to drag and drop it, creating personalized unique displays.

Example XML files are found in the next page, and show how these views are generically created. In addition, though the screen has simple buttons but this functionality will be moved to an expanding tabs settings menu from top right.

This is the bar layout XML file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="250dp"
   tools:context="com.lfev2017.ktdilsiz.cellapptest.DisplayGrafActivity">
   <com.github.mikephil.charting.charts.BarChart
      android:layout_width="match_parent"
      android:layout_height="200dp"
      android:id="@+id/bargraph" />
   <TextView
       android:layout_width="match_parent"
       android:layout_height="50dp"
       android:text="@string/hold_me_here"
       android:layout_below="@id/bargraph"
       android:gravity="center"/>
</RelativeLayout>
```

As you can see, this is simply created through an id of “bargraph” and connects to the third party library we are utilizing. To populate the graph, you simply pass the values through our BarChartGenerator class and it displays the data.

```
View myLayout = inflater.inflate(R.layoutbar_layout, mainLayout, false);
mainLayout.addView(myLayout);
```

This is how simple it is to be able to dynamically add the views.

In the next few pages, we will have some examples of how the user can create some specific exemplary charts/graphs.

Figure 9. Single Pack vs Time display -- Here you can see that the time is adjusted to display the data from every hour. And this graph is generated through Accumulator Chart generation which is explained in figure 2.3. Only Pack 4 is shown here but it can be adjusted as the whole accumulator or even a single cell.

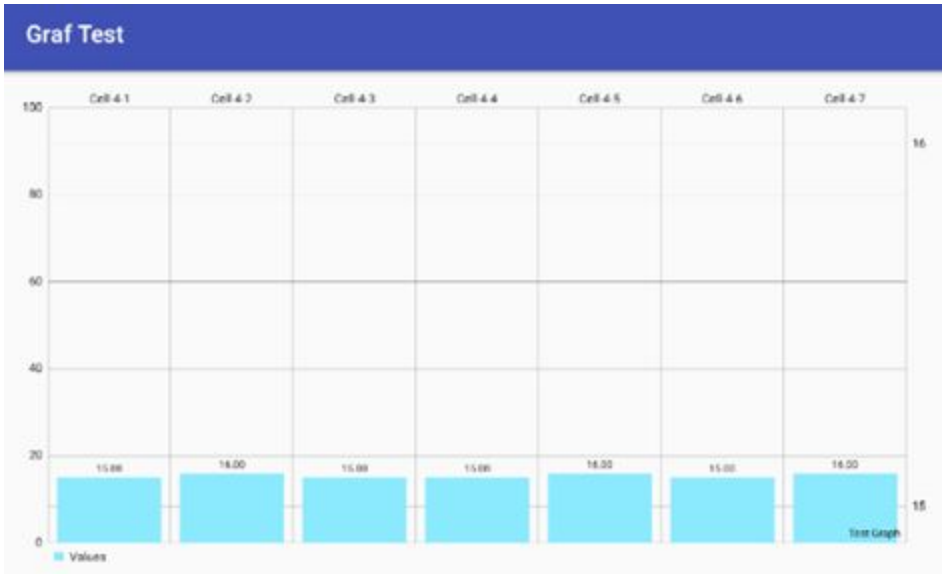


Figure 10. 7 Cells are displayed at the same time in a pack. Here we are looking inside the Pack 4 and each cell has a data that is updated every time period specified by the user from the web-server. This BarChart shows how the chart can be used easily to compare the cells to each

other. Also if you compare this figure to Figure 5.2, you can see how Bar Chart can be used both against time for getting an update and also amongst the parts of the system for easy comparison.

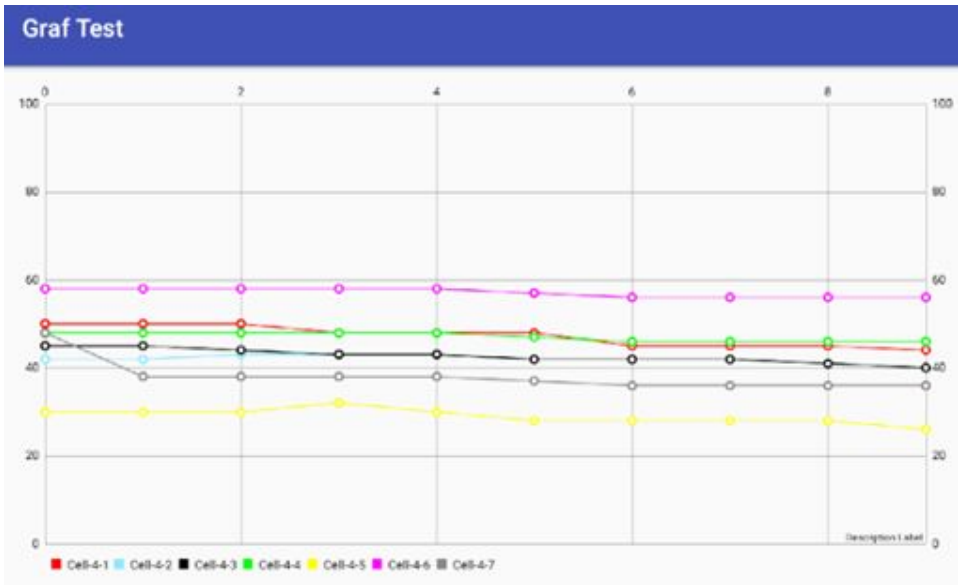


Figure 11. Cells Line Chart -- Again similar to figure 4.3, here we are looking at the seven cells from pack 4. However, this time we can see the data in a Vs time LineChart and combine the functionality of comparing cells with the functionality of seeing time update. This again is an example of accumulator chart generation from figure 2.3.

Figure 12. Emulator View Horizontal for Tablet emulator

Another important aspect of the user created charts, views is how the user will be able to access this data and how easily they can navigate through the information. Under Data Structures section, we have talked about how the data structures will be created with the names of the sections. Such as “tsvcell1voltage”. A similar method will be used for creating an “expanding tabs” interface for easily choosing the right data.

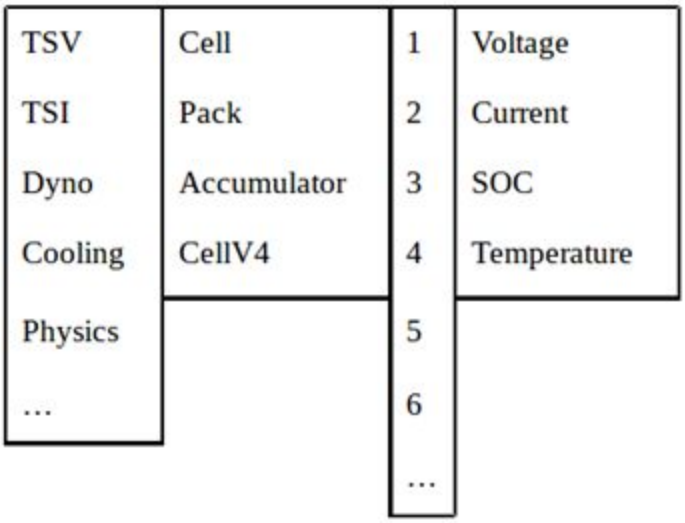


Figure 13. Here you can observe that the menus will be opening one by one through the selections. “TSV” → “Cell” → “1” → “Voltage” will create the hashcode of “tsvcell1voltage” and refer to the necessary hashmap created by the application.

We are also planning to implement a ViewPager into the application to be able to multiple displays that will have the dynamically inflating views capability.

Figure 14. This will create a display similar to how Android home pages are formed. You can basically choose a display, drag and drop it anywhere, zoom in or out and even save your displays. This is to be able to create a fully customizable experience.

Testing Protocol

Testing will be done through Android Studio standards.

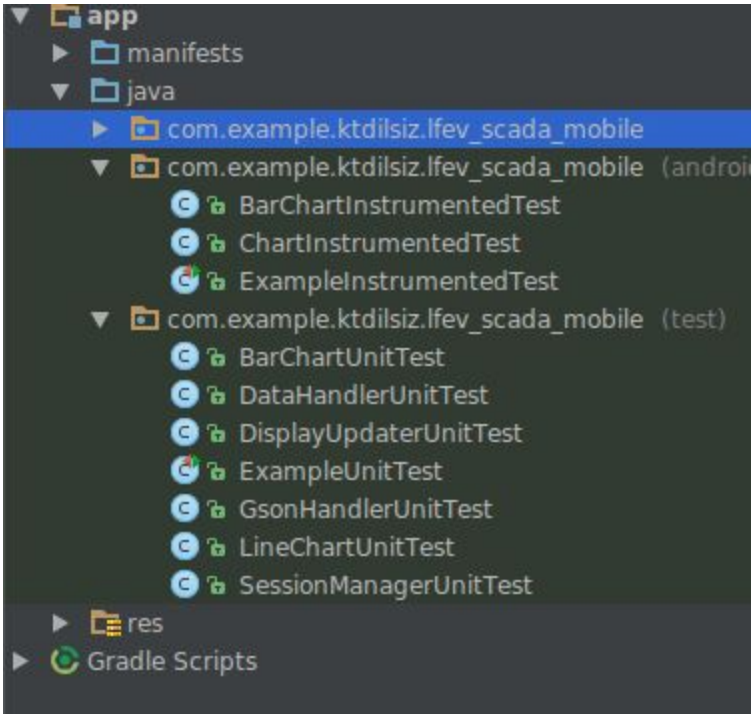


Figure 15. Here you can see how we have created separate unit test files for each class. There will be two kinds of tests, "InstrumentedTests" and "UnitTests".

Unit Tests run on the local JVM and Instrumented Tests run on the Android device. For more information please visit: https://developer.android.com/training/testing/start/index.html

Example Unit Test Code:

```
/**
* Example local unit test, which will execute on the development machine
(host).
*
* @see <a href="http://d.android.com/tools/testing">Testing
documentation</a>
*/
public class ExampleUnitTest {
   @Test
   public void addition_isCorrect() throws Exception {
      assertEquals(4, 2 + 2);
   }
}
```

Documentation Protocol

For documenting our methods and classes, we will use Javadoc structure.

```
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute{@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists. When this applet attempts to draw the image on
 * the screen, the data will be loaded. The graphics primitives
 * that draw the image will incrementally paint on the screen.
 *
 * @param  url  an absolute URL giving the base location of the image
 * @param  name the location of the image, relative to the url argument
 * @return  the image at the specified URL
 * @see     Image
 */
 public Image getImage(URL url, String name) {
      try {
            return getImage(new URL(url, name));
      } catch (MalformedURLException e) {
            return null;
      }
 }
```

Example 7.1: Taken from: http://www.oracle.com/technetwork/articles/java/index-137868.html

Here as you can see javadoc requires the explanation for the method, the parameters, the return object and the references for the code. This generates a very nice javadoc documentation that can be easily converted into an API style HTML source.

This documentation will make the project sustainability much easier and also make the application much more accessible to the future teams for further expandability.

Conclusion

Currently, the design has some major structure laid out and we are following these plans to be able to create a product that is not only customizable but also maintainable and expandable.

We have looked into many different aspects of the application such as how the data structures will be implemented and how the JSON parsing will be structures by the SCADA team.

In addition, we were able to observe how the layout is going to be implemented although it is in a very primitive stage and we were able to see few examples of charts that could be implemented through the planned application. We have talked about ViewPager and how we will be able to slide in between different displays/fragments of displays.

The testing and the documentation is the standard protocol and we will be using the most common tools that are easy to implement and maintain in the system. This makes it possible to ensure that the future teams can easily update or maintain the system. An example would be the usage of the most popular third party library for creating charts although implementation of another one might be easier.

Also the activity diagrams and the development view provides a general overview of the planned system. Currently, we are also working on creating a detailed UML diagram including every class, every method and every connection in the system but this is not yet completed.

Lastly, although every feature is important and essential to the application. We had to make some priority selections among the tasks to be completed. This is an example of how we have some trade offs in the system to be able to complete an application that is of the most satisfactory for the LFEV team of 2017 and for the future.

Tractive System Interface (TSI)

Brakes

Brake Overtravel Switch

The brake overtravel switch is a manual reset limit switch that will actuate a normally open relay in the safety loop. Voltage and signal will be sent and received from the TSI box.

Brake Pressed Signal

The brake pressed signal comes from a pressure-actuated switch in the pedal cluster. This is treated like a normally open switch, and will be brought into the PCB and microcontroller to be used in throttle control and system state decisions.

Brake Light

The brake light is controlled by the brake pressed signal, and is displayed using a 12V, 74 cm 2 golf cart red brake light compliant with DOT FMVSS 108 as a brake light (as required by the Formula Hybrid Rules T7.4).

Throttle

Accelerator Pedal Plausibility System (APPS)

The APPS consist of two piston potentiometers connected to the throttle pedal. These two potentiometers are offset from one another by 5V, with their wiper outputs used in a series of checks used to ensure that a valid throttle value is sent to the motor controller. For one check, the two signals are put through a differential op-amp, followed by a window comparator in order to confirm the two signals remain within 10% of one another (0.5V). Separate checks are made on each APPS signal individually, sending each signal through two other window comparators, making sure that the signals are not open/short-circuited.

Throttle to Motor Controller

The throttle to the motor controller is sent from the low voltage side of the board to the high voltage side by means of an isolated differential amplifier. This is a 0-5V signal fed directly into the motor controller. On the low-voltage side, the throttle can be sourced from one of twolocations. First, the analog voltage passed from the APPS may be sent through an analog switch. The second option, which facilitates VSCADA control, is a PWM signal sent from the microcontroller which is low-pass filtered (integrator with time constant of 1.6ms) then sent to the switch. Selection of the throttle is done by the microcontroller. This gives the microcontroller

the ability to cut the throttle to the motor controller, by outputting 0V and selecting the microcontroller output on the analog switch.

Acceptance of VSCADA Throttle Control

TSI has the ability to accept VSCADA throttle control over the CAN bus. VSCADA may request to increase or decrease speed, which the TSI microcontroller may do by altering the PWM throttle output. This allows the TSI to maintain safety checks, while still allowing VSCADA to remotely request control of the throttle.

IMD

Connected to Safety Loop

The IMD we are using is the Bender ISOMETER IR155-3204 to meet Formula Hybrid Rule EV7.9.2. This has two signal coming out of it that will be used for interacting with the safety loop
and TSI system. One of these is the OK HS high or low voltage. Under normal operating conditions, this is set to 24V as this is the GLV power the board will be receiving. If the IMD senses the high voltage connections have connected to low voltage ground, this signal will change to 0V. We will be using this to control the IMD safety loop relay.
The other signal we are using from the IMD is the PWM output of the measured resistance. This will be sent through a low pass filter and into the microcontroller. It will then be decoded to determine the resistance based on the duty cycle. This value will be sent over CAN to the SCADA team.

IMD Fault Light

The IMD fault light will activate when the IMD is the reason the safety loop is broken. This will be achieved by using the second output from the IMD relay used for the safety loop. The relay we are using has two outputs based on the control signal so if the OK_{HS} goes to 0V, the output of the relay will flow to the fault light instead of the safety loop.

Voltage and Current Measuring

Read Voltage and Current Values from TSV

To read the current provided from the TSV system, we will be using an Ametes BBM-01 current sensor mounted to the metal power bar inside of the TSI box. For the voltage, we will be using an isolated differential op-amp.

HIgh Voltage Present Lights

The high voltage present lights will be activated by a DC-DC convertor. The TSV voltage will go through buck converter, and will be scaled down to 24V. The buck converter will be set to have an on voltage of 30V. The 24V will go to an isolated DC/DC converter that will scale it down to 12V, which will be used to power the lights.

TSMP

The TSMP will be mounted to the front of the TSI box. The inputs to each of the banana jacks will be taken from the metal bars running through the box. At the beginning of the cables going from the bars to the measuring points, a 10k ohm resistor will be added in series with a power rating of 5W to meet Formula Hybrid rule EV10.3.5.
The third banana jack will be connected to the chassis ground provided to the TSI box for intentionally causing a ground fault in the IMD.

Shutdown System

State Diagram Compliant with Formula Hybrid Rules

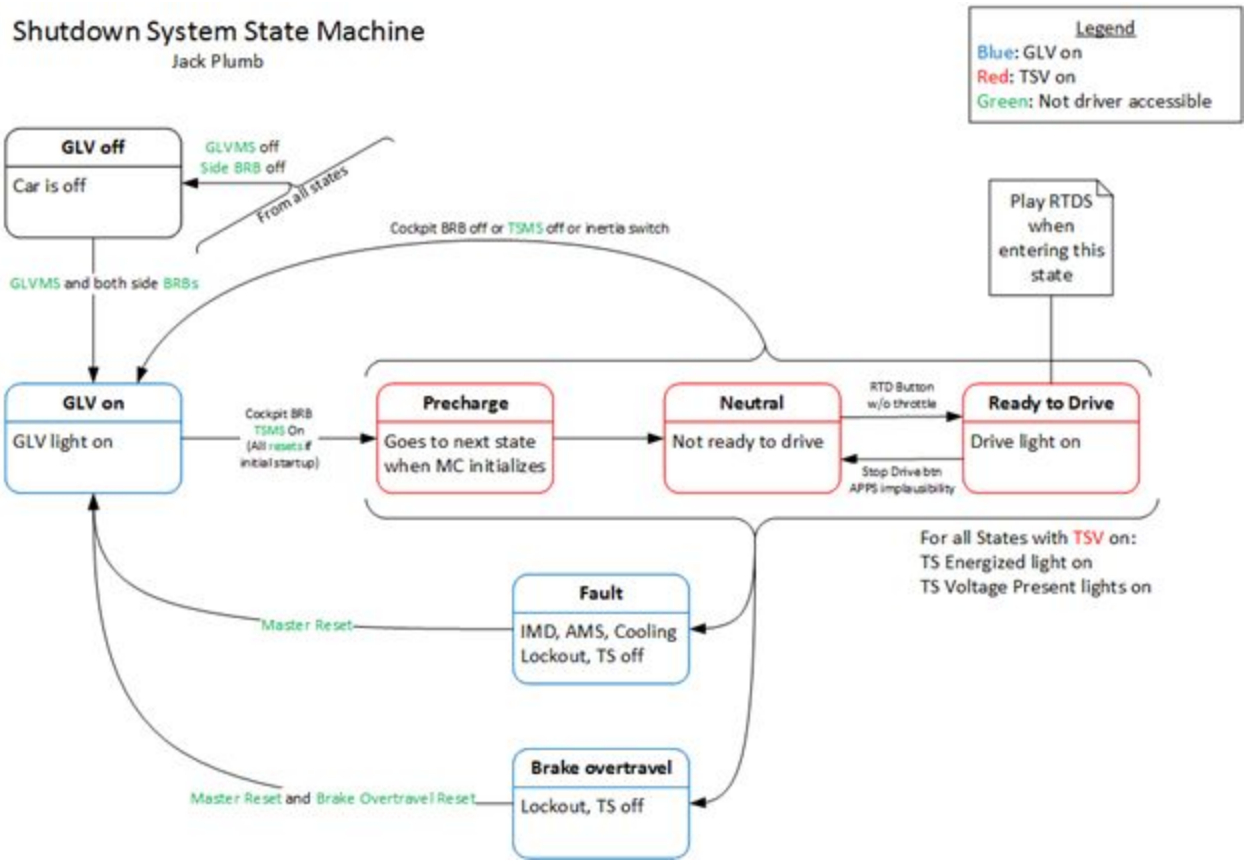


Figure 16. State Diagram for Shutdown.

Ready to Drive Button

The ready to drive button will be mounted on the dashboard and accessible to the driver. This is the last step in the startup sequence. In order for the button press to be accepted, the driver must have his/her foot on the brake so the car will not move on startup.

RTDS

The RTDS will activate when the drive button is pressed and the microcontroller is in the state to allow for driver control. It will sound for 2 seconds to meet Formula Hybrid rule EV9.2.1.

Drive Light

The drive light will be active after the drive button is pressed and the RTDS activates. This light will be mounted on the dashboard to let the driver know they are able to move the car. This will be controlled by the microcontroller that is in charge of the startup sequence.

Motor Controller

FWD/REV Key

A single-pull double-throw keyswitch will be used to select forward or reverse motion of the motor controller. This will be mounted on the TSI box, out of reach of the driver. Voltage will be supplied by the motor controller, with the signal sent directly back into the motor controller.

Precharge Relay

The precharge relay is automatically actuated by the motor controller after the internal start-up sequence when TSV is applied.

Interface via CAN Bus/Control System Cable

The motor controller is connected to the system-wide CAN bus via a CAN bus isolator. This allows motor controller information to be sent directly to VSCADA.

Grounded Low Voltage (GLV)

Overview

The GLV system is responsible for managing the low voltage electronics of the vehicle. The four main purposes of GLV are 1) providing 24V to all the main sub-systems of the vehicle, 2) providing a physical user interface for the car, 3) providing a means for the sub-systems to interface with the VSCADA computer and 4) providing a safety system to shut down systems in the car. We have broken these four main purposes down into four main sub-systems of GLV, respectively called GLV Power, Vehicle User Interface (VUI), Vehicle Computer Interface (VCI) and the Safety Loop. We will now explain the general requirements for each of the four GLV sub-systems.

The GLV Power system should deliver a 24V to all systems in the car and should be rechargeable by means of a UL listed charging device.

The VUI includes physical panels, buttons, switches, screens and lights on the car, such as the exterior panels and the cockpit panel. The wiring for these panels and how they interface with the system is managed by GLV.

The VCI should deliver information about GLV Power and the Safety Loop to the VSCADA computer and route the CANBUS to the VSCADA computer. Any other information that needs to be relayed to the computer is managed by VCI.

The Safety Loop should serve as the direct means to control the Accumulator Isolation Relays in the packs (enable/disable of high voltage systems). Under unsafe conditions, the Safety Loop should make sure that the AIRs are closed.

All of the sub-systems have significant interdependencies. The detailed design of their integration will be explained in the next section.

Design

In this section of GLV, we will discuss our design goals for this year, explain high level design of our system, show proposed physical implementation of the systems and detail subsystem functionality.

Our design goals for this year were to consolidate aspects from last year's design into a cleaner form and add more feature to meet all the requirements of GLV. To do this, we decided to move all GLV functionality into a single box.

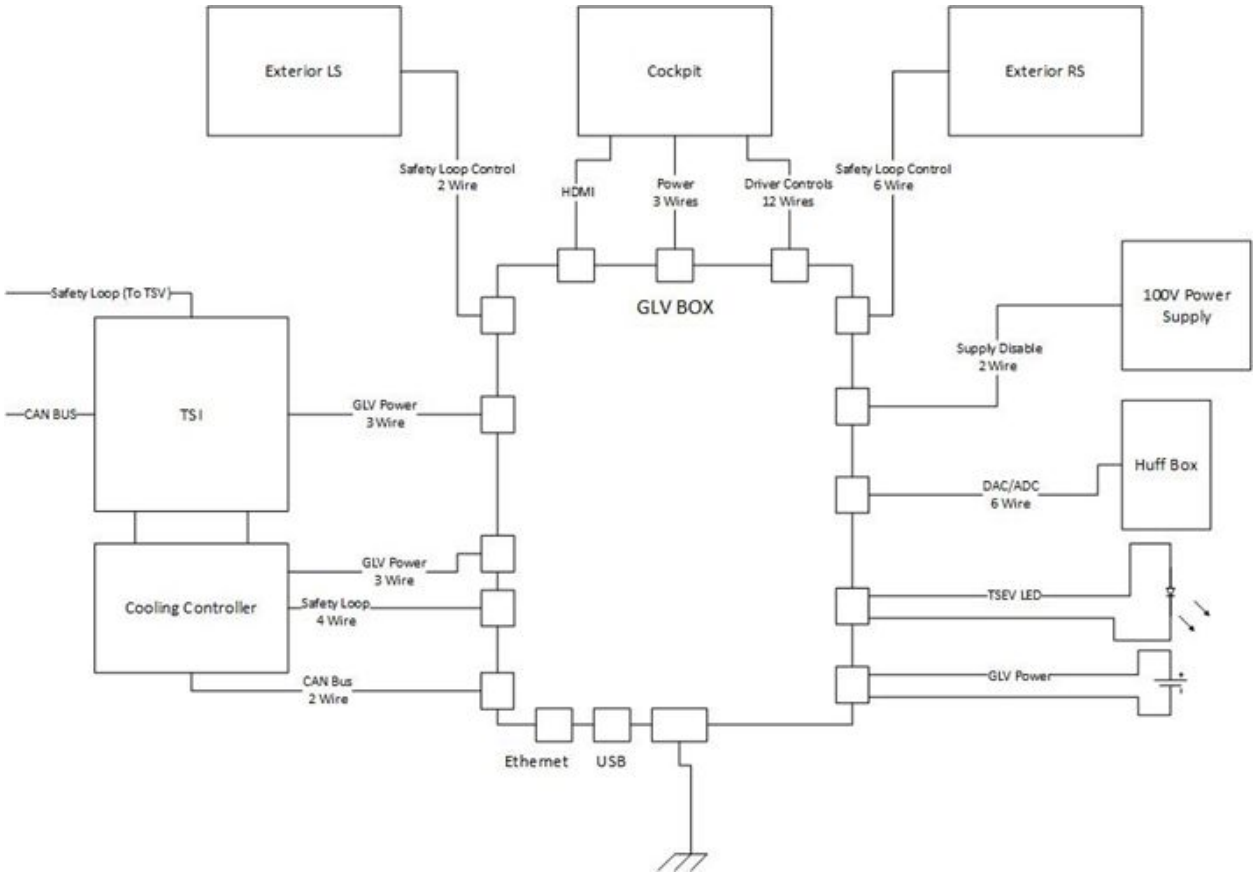


Figure 17. High Level GLV System Connections.

As shown in Figure 17 above, the GLV Box now has all of its features self-contained and connections between most external systems have been consolidated into a single connection. Inside of the GLV Box is a BoB managing Safety Loop, VCI and GLV Power functionality, a Raspberry Pi (VSCADA), and some connectors between the connectors and boards.

On the outside of the box, we expect to create a connector panel that can be mounted over a hole in the box. Below in Figure 18, is a proposed layout for the connector panel of the box. Most of the connections can be accomplished by using Deutsch DT-04 connectors.

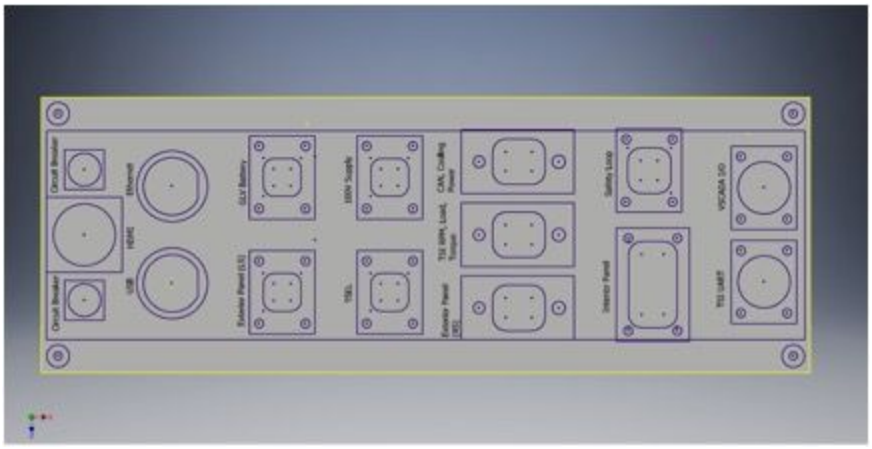


Figure 18. Connector Panel for GLV Box (subject to revisions).

Below in Figure 19, is our proposed layout of the system inside the GLV Box. The dark green box represents the Raspberry Pi, the light green box represents the GLV BoB, the white box represents the CAN2USB adapter, and the light gray box represents the DIN rail terminals.

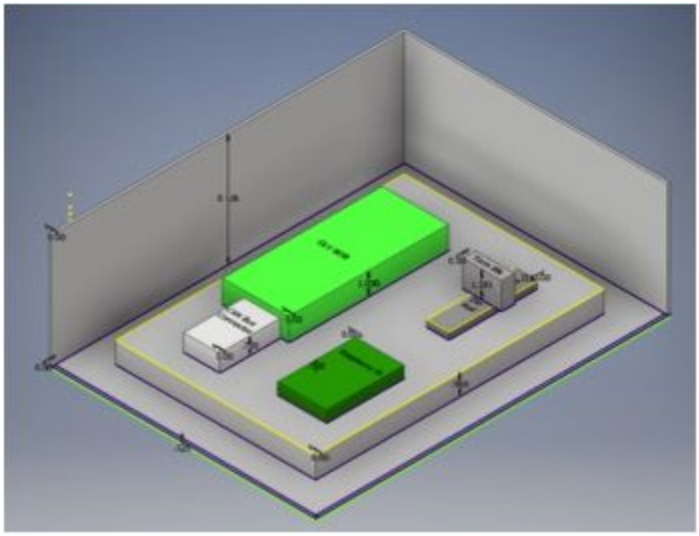


Figure 19. GLV Box Layout (subject to revisions).

GLV BoB

Most GLV functionality is contained on the BoB inside of the GLV box. This board is a critical component of our design and we had to make some design choices in considering how it would be implemented.

Old GLV/VSCADA/Safety BoB

Last year's team printed a PCB that managed aspects of the Safety Loop, VCI and GLV Power. While their approach made sense and was a good start, it was not comprehensive in meeting all

the requirements of the GLV system and did not encapsulate as much of the system as we desired. We considered using this board in this year's design to save time and money; however the design we implemented left out some key VCI components and was very messy in terms of interconnect. We since have decided to create a new GLV BoB but have kept our design with the old GLV BoB as a design alternative.

New GLV BoB

The main tasks we had in creating a new BoB were adding/simplifying VCI components and making wiring and interconnect simpler for the whole GLV system. In order to do this we added more routing to the board so that a larger portion of the Safety Loop is contained on the BoB. In addition, we added Safety Loop and SOC monitoring systems to interface with the VSCADA computer. While this board will likely have more screw terminals and be larger, it will be more effective and simplify our design. In Figure 20 below, a high level block diagram for the new GLV BoB is shown.

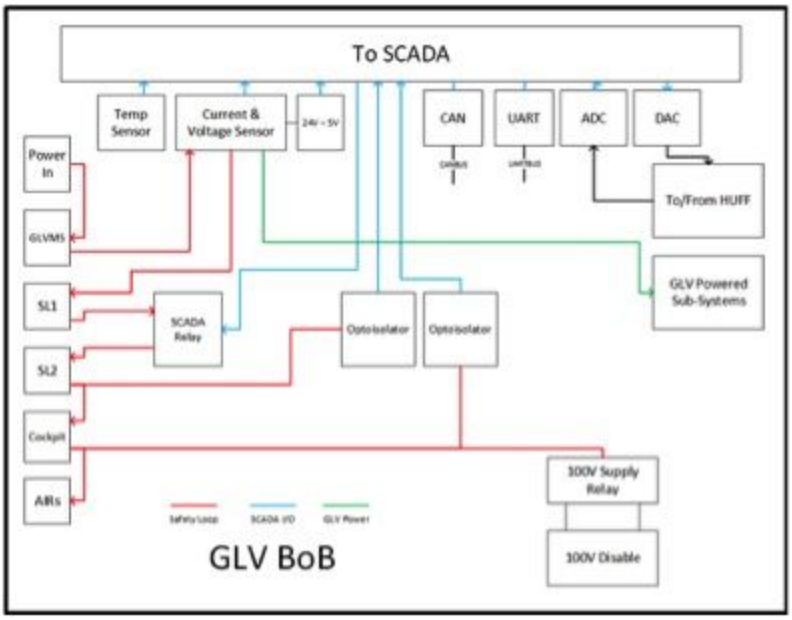


Figure 20. High Level GLV BoB Block Diagram.

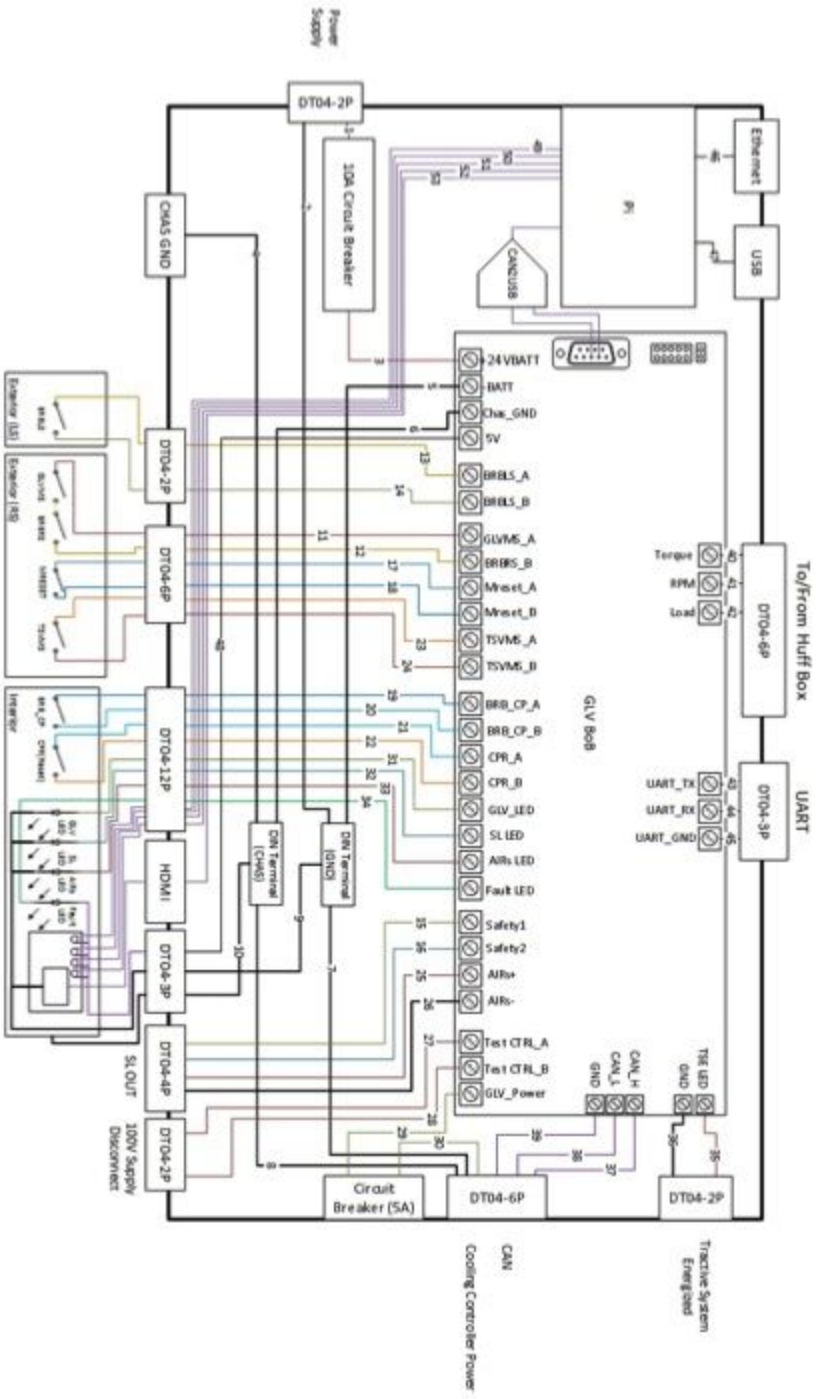


Figure 21.

In Figure 21 above, you can see how the full system would be implemented inside of the GLV Box with the new GLV BoB on a detailed level. This design documents each wire that would be found inside of the box and a tentative wire list for the implementation can be found below:

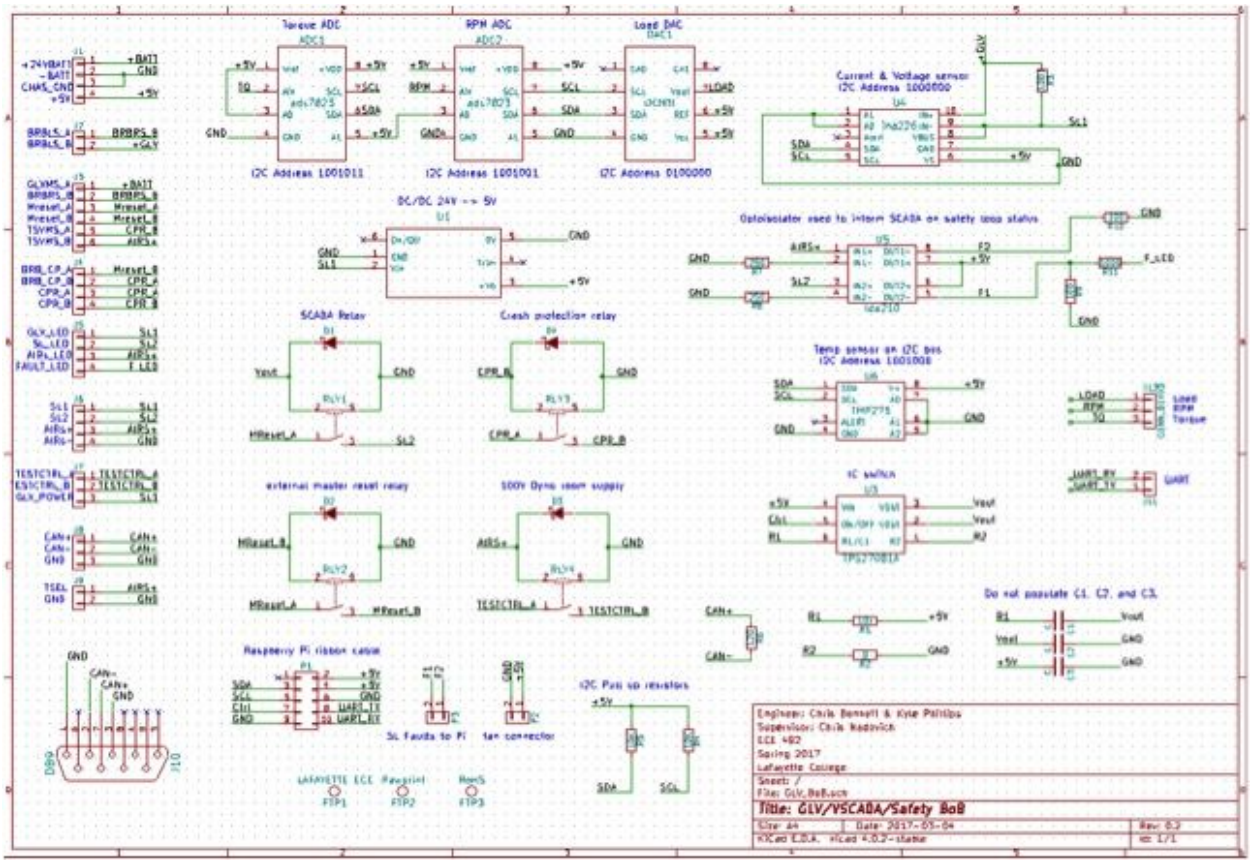


Figure 22.

Wire Number	Conn A	Conn B
1	Battery +	High Amp Circuit Breaker In
2	Battery -	DIN Terminal (GND)
3	High Amp Circuit Breaker	BoB (+24VBatt)
4	CHAS GND	DIN Terminal (CHAS)
5	BoB (-BATT)	DIN Terminal (GND)
6	BoB (Chas_GND)	DIN Terminal (CHAS)
7	DIN Terminal (GND)	GLV Power Out (GND)
8	DIN Terminal (CHAS)	GLV Power Out (CHAS)
9	DIN Terminal (GND)	Int Panel Power (GND)

10	DIN Terminal (CHAS)	Int Panel Power (CHAS)
11	BoB (GLVMS_A)	Ext Panel RS (GLVMS)
12	Ext Panel RS (BRBRS)	BoB (BRBRS_B)
13	BoB (BRBLS_A)	Ext Panel LS
14	Ext Panel LS	BoB (BRBLS_B)
15	BoB (Safety1)	SL OUT (SL1)
16	SL OUT (SL2)	BoB (Safety2)
17	BoB (Mreset_A)	Ext Panel RS (Mreset)
18	Ext Panel RS (Mreset)	BoB (Mreset_B)
19	BoB (BRB_CP_A)	Int Panel (BRB_CP)
20	Int Panel (BRB_CP)	BoB (BRB_CP_B)
21	BoB (CPR_A)	Int Panel (CPR)
22	Int Panel (CPR)	BoB (CPR_B)
23	BoB (TSVMS_A)	Ext Panel RS (TSVMS)
24	Ext Panel RS (TSVMS)	BoB (TSVMS_B)
25	BoB (AIRs+)	SL OUT (AIRs+)
26	BoB (AIRs-)	SL OUT (AIRs-)
27	BoB (Test_CTRL_A)	100V Supply Disconnect A
28	100V Supply Disconnect A	BoB (Test_CTRL_B)
29	BoB (GLV_Power)	Low Amp Circuit Breaker In
30	Low Amp Circuit Breaker Out	GLV Power (24V)
31	BoB (GLV_LED)	Int Panel (GLV_LED)

32	BoB (SL_LED)	Int Panel (SL_LED)
33	BoB (AIRs LED)	Int Panel (AIRs LED)
34	BoB (Fault LED)	Int Panel (Fault LED)
35	BoB (TSE LED)	TSE Light (+)
36	BoB (GND)	TSE Light (-)
37	BoB (CAN_H)	CAN (H)
38	BoB (CAN_L)	CAN (L)
39	BoB (GND)	CAN (GND)
40	From HUFF (Torque)	BoB (Torque)
41	From HUFF (RPM)	BoB (RPM)
42	From HUFF (Load)	BoB (Load)
43	BoB (UART_TX)	UART (TX)
44	BoB (UART_RX)	UART (RX)
45	BoB (GND)	UART (GND)
46	Ethernet	Pi (Ethernet)
47	USB	Pi (USB)
48	BoB (+5V)	Int Panel Power (5V)
49	Pi (I/O)	Int Panel (Buttons)
50	Pi (I/O)	Int Panel (Buttons)
51	Pi (I/O)	Int Panel (Buttons)
52	Pi (I/O)	Int Panel (Buttons)
53	Pi (HDMI)	Int Panel (HDMI
54	BoB (10 Pin Header)	Pi pins

55	BoB (2 Pin Header)	Pi pins

We will now provide a more detailed explanation for how each of the sub-system functionality is being implemented.

GLV Power

We have decided the best way to deliver 24V to all systems on the car is with a 24V Lithium Ion Phosphate ($LiFePO_4$) Battery. Since our battery is commercially manufactured, plug-an-forget capability and protection against over-voltage, over-current and over-charge is already implemented in the battery as a Battery Management System (BMS).

The battery will be housed in its own separate box and interfaces directly with the GLV box with a two wire (Power, GND) cable. Power should be delivered to the subsystems after the GLV master switch is thrown and the two exterior BRBs are closed.

Another important aspect of GLV Power is the max and nominal current draw from the system. The max current draw from opening the AIRs we have measured at 4.6A. The rest of the current will be drawn from TSI, the cooling controller, VSCADA, and the LEDs, relays and displays. TSI has estimated a 2A max pull and Cooling Controller is estimated to pull 5A max. This gives us a possible max current pull of 12A. Thus the first circuit breaker placed directly after the battery is 15A, which is at least 25% more than the expected max possible current in our design. The second placed directly before the TSI and Cooling Controller power lines we have set at 8A, also 25% more than the expected max possible current draw. These numbers and choices are subject to change as we get more accurate estimates for current draw.

The nominal current draw to keep the AIRs open is .5A. The nominal current draw from TSI and Cooling Controller has been estimated at 2.5A. This gives an overall nominal current draw of at least 3A. Under these conditions, our 10Ah battery, should be able to last a little less than a 3 hours.

Safety Loop

The basic design of the safety loop was taken from Y4 (2016). See Figure # below:

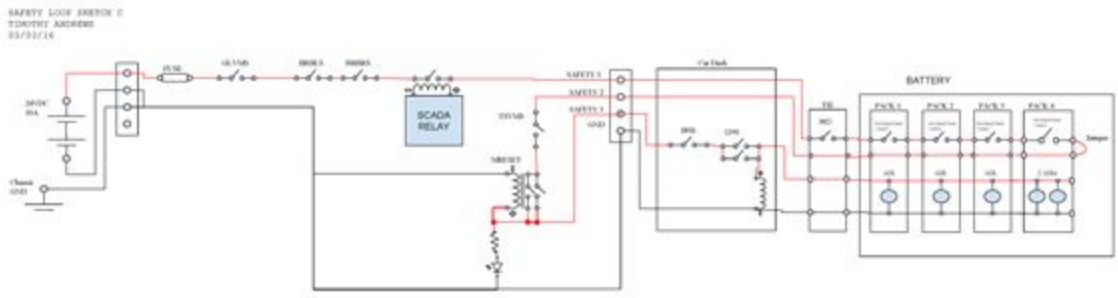


Figure 23. Safety Loop Diagram from 2016.

The safety loop requires the flipping of several switches and the safety status of the TSI and battery packs to be OK in order to power the AIRs. In addition to this, VSCADA can control a relay which allows it to trip the Safety Loop at any point. We have kept a lot of this design the same but have made a couple important additions. See Figure # below:

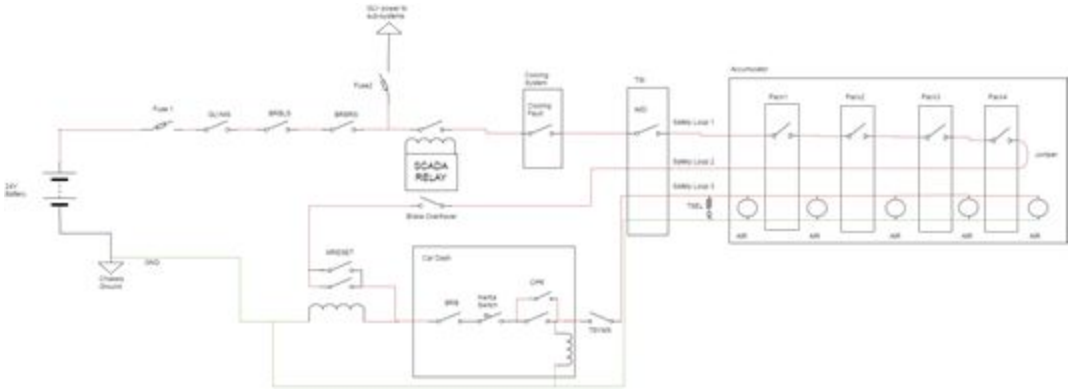


Figure 24. Safety Loop Diagram From 2017.

In specific, we added the cooling controller to the safety loop, the point where GLV sub-systems are powered up and moved the TSVMS to the point directly before the AIRs. Other than this, the Safety Loop design is exactly the same as last years.

We have moved the routing of the safety loop on to the new GLV BoB and have provided connections on the GLV Box to all of the control panels for the Safety Loop.

Vehicle User Interface

The VUI design can be thought of as the three main panels on the car, the left side exterior panel, the right side exterior panel and the interior panel. The two exterior panels provide controls for interacting with the start-up and shut-down procedures of the car, i.e. the Safety Loop.

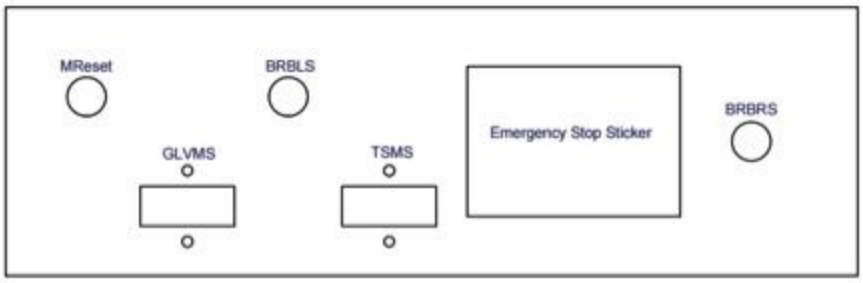


Figure 25. Exterior Panel Combining LS and RS for Dyno Room.

The interior panels includes buttons to control the start up the car and reset the safety loop, LEDs to indicate the status of the car, and the VSCADA I/O. The VSCADA I/O includes an LCD screen and three buttons for the driver to move through the display and initiate cruise control.

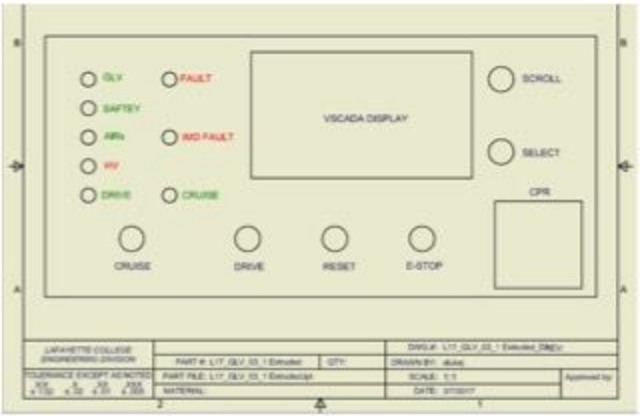


Figure 26. Interior Panel for Dyno Room

Vehicle Computer Interface

The VCI system implementation is heavily integrated with the other sub-systems. All of the VCI functionality has been worked onto the new GLV BoB and we have added several new IC's on the board in order to achieve the requirements for the system.

Safety Loop Monitoring

In order to deliver the status of the Safety Loop to the VSCADA computer we have decided to use optoisolators to monitor the presence of 24V at different points in the safety loop. When different parts of the Safety Loop fail, we will be able to detect which points are still receiving 24V and then give 5V high or low signal to the VSCADA computer indicating such. In our design we have chosen to detect the status of the Safety Loop after going through all of the other sub-systems on the car and at the point which the AIRs are powered.

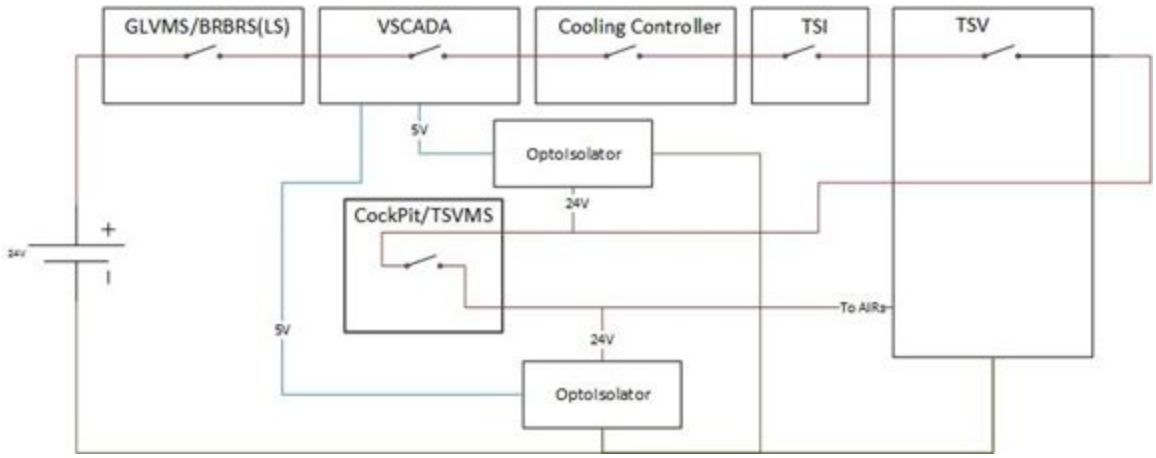


Figure 27. Optoisolator connections for safety loop monitoring.

With this design VSCADA is able to know when there is an OK status from all the sub-systems and when the AIRs are powered.

State of Charge Monitoring

In order to deliver the SOC of the GLV battery to VSCADA, we are using the INA226 current shunt and power monitor to talk over I^2C to the computer. Using this chip we can constantly keep the computer updated on the total GLV current and GLV SOC. This chip is placed on the GLV BoB.

CAN Bus Routing

The routing of the CANBUS is achieved with a two pin connector into the GLV box. On the GLV board, the differential traces of the CAN_H and CAN_L signal are sent to the CAN2USB adapter connected to the GLV board.

Torque, RPM Reading and Load Control

There are two ADC's on board the BoB to read incoming analog signals from the Huff box about Torque and RPM. There is also a DAC on the BoB to allow VSCADA to set the Load on the Huff Box.

Motor Controller Cooling System

Overview

The 24VDC powered motor controller cooling system is able to cool the motor controller based on temperature sensors' readings, to provide system status (such as fan RPM, coolant flow speed and sensor readings) to SCADA via CAN bus and to interface with LFEV safety loop with a fail safe design.

Designed Details

The whole cooling system works on 24VDC provided by GLV team and the core system controller is one Arduino Uno. The system controller will have the ability to measure the inlet, outlet fluid temperature, and fluid flow rate and these data will be accessible over the CAN bus (by using Arduino CAN bus shield) and will be shown on a LCD display connected to the system controller; it will also have the ability to control the fan and pump speed automatically based on fluid temperature (parameters for the control algorithm, as well as manual override, shall be possible); moreover, it will be able to interface with the LFEV safety loop and contain a fail-safe design that will open the safety loop when when fluid temperature is too high.

System Connection Diagram

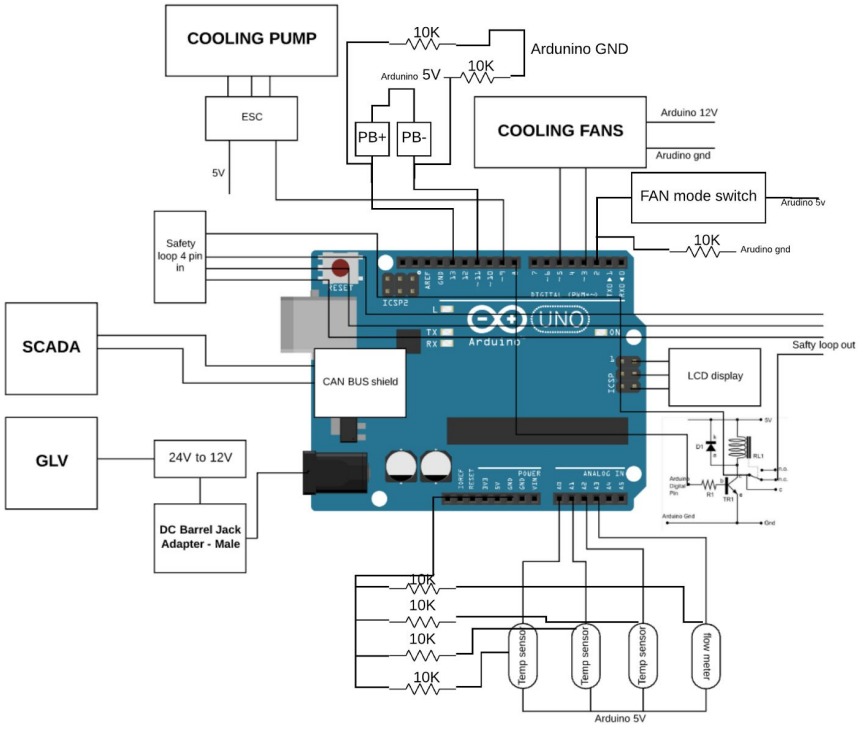


Figure 28.

Coolant Flow Diagram

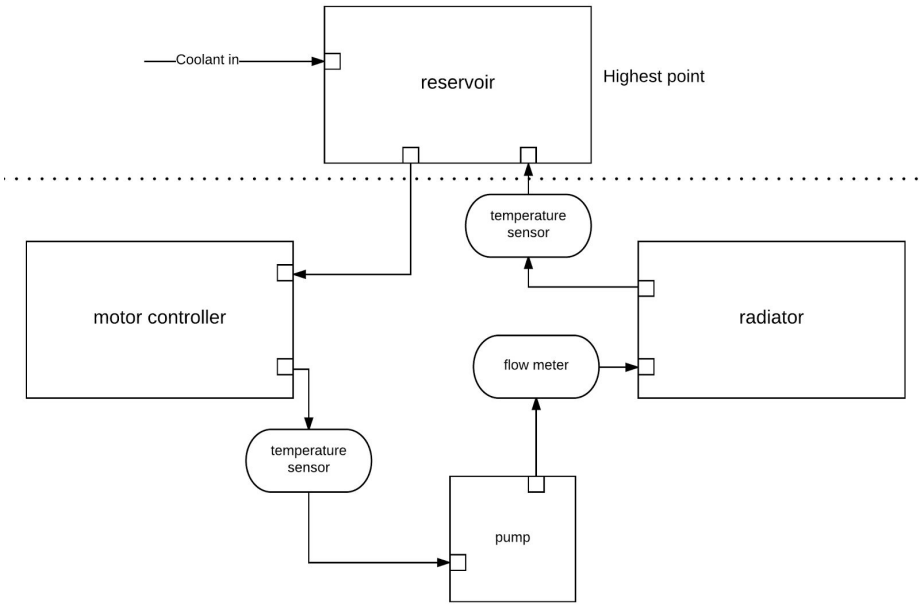


Figure 29.

Compatibility and Independence

The same as the old cooling system, 13x19mm PVC tubing is used across the water cooling system as well as Koolance QD4 disconnects and other connectors. To the largest extent possible the system is using components purchased from Koolance that are identical to or compatible with the existing cooling system, so testing of the water cooling system can be easily done by replacing the old system with this new one. Though the new cooling system will be independent that can run on its own. Also, the water pump will be compatible with the 24VDC provided by team GLV.

Automatic Fan/Pump Speed Control with Manual Override

Based on temperature readings on the motor controller and temperature readings on coolant, the fan and pump speed are automatically controlled by Arduino Uno to keep the motor controller cool based on specified temperature thresholds, which can also be modified by using push buttons and switches connected to Arduino without re-programming the board.

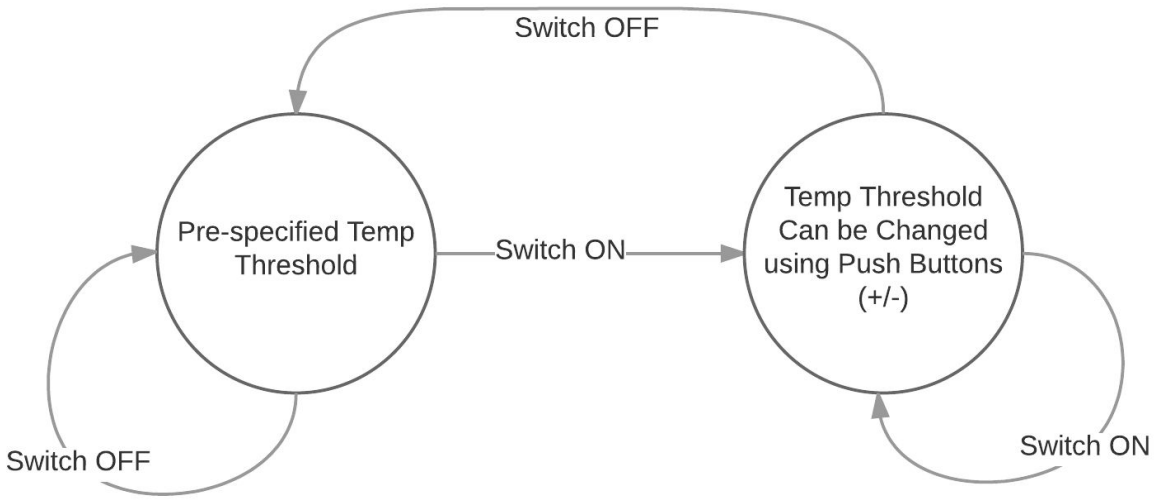


Figure 30. Diagram showing the threshold overriding process

Can Bus Interface

The CAN bus interfacing will be done using a stackable SparkFun CAN bus shield using SPI interface on Arduino Uno. Raw CAN_HI, CAN _LOW and GND signal will be extracted from the standard 9-way sub-D port on the CAN bus shield and will be fed into SCADA software. The CAN signal can also be tested by displaying the CAN_HI and CAN_LOW signal on oscilloscope.

Safety Loop Interface

The cooling system is also connected to the 4-wire LFEV safety loop, when the coolant temperature is too high the cooling system should shut the system down by opening the safety loop.

Acceptance Test Plan

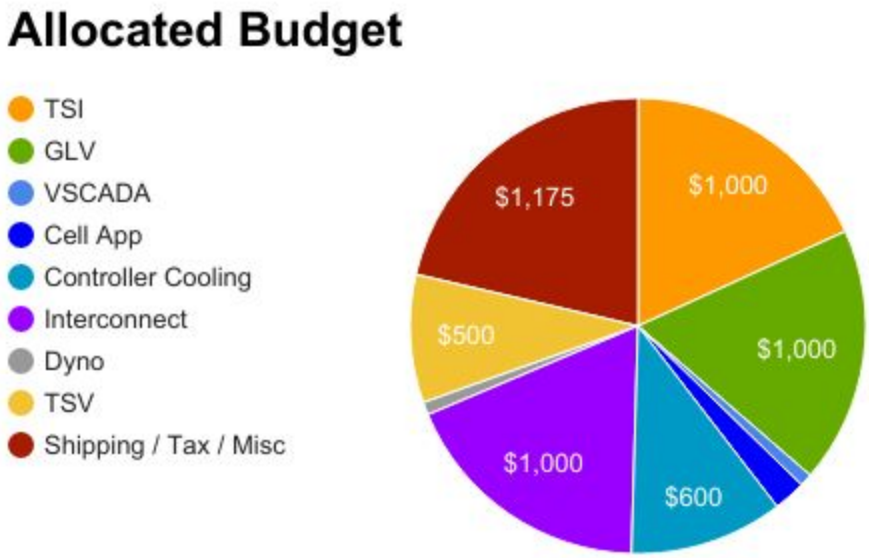
See ATP-Overview for information about system testing.
For information about subsystem QA testing click here

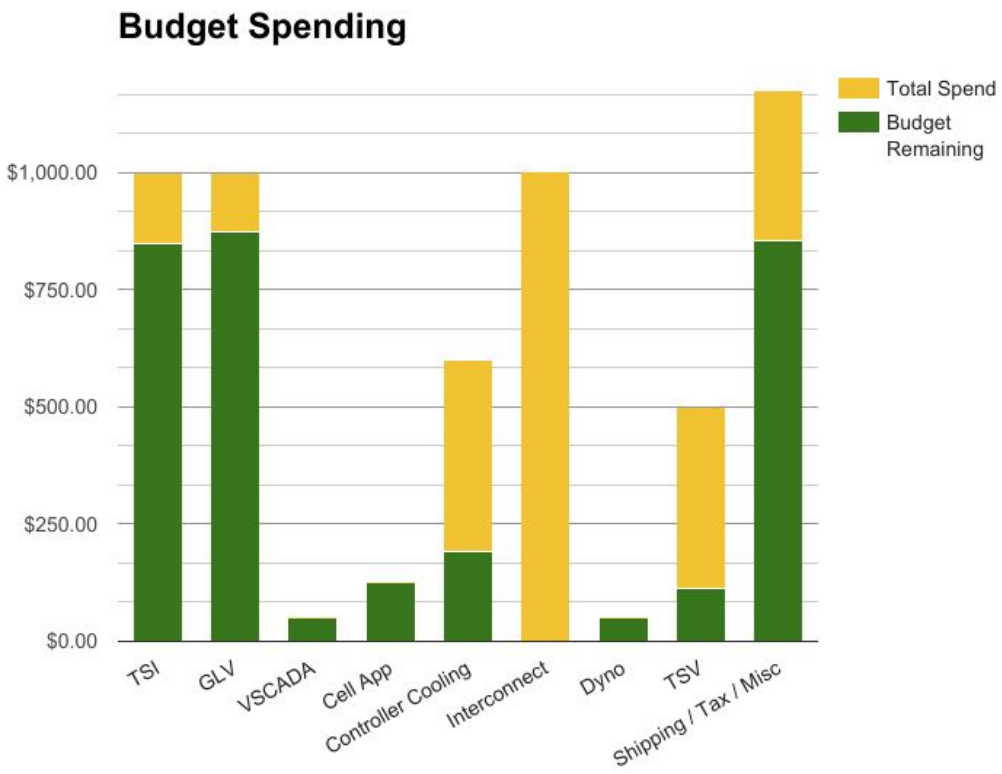
Cost Analysis

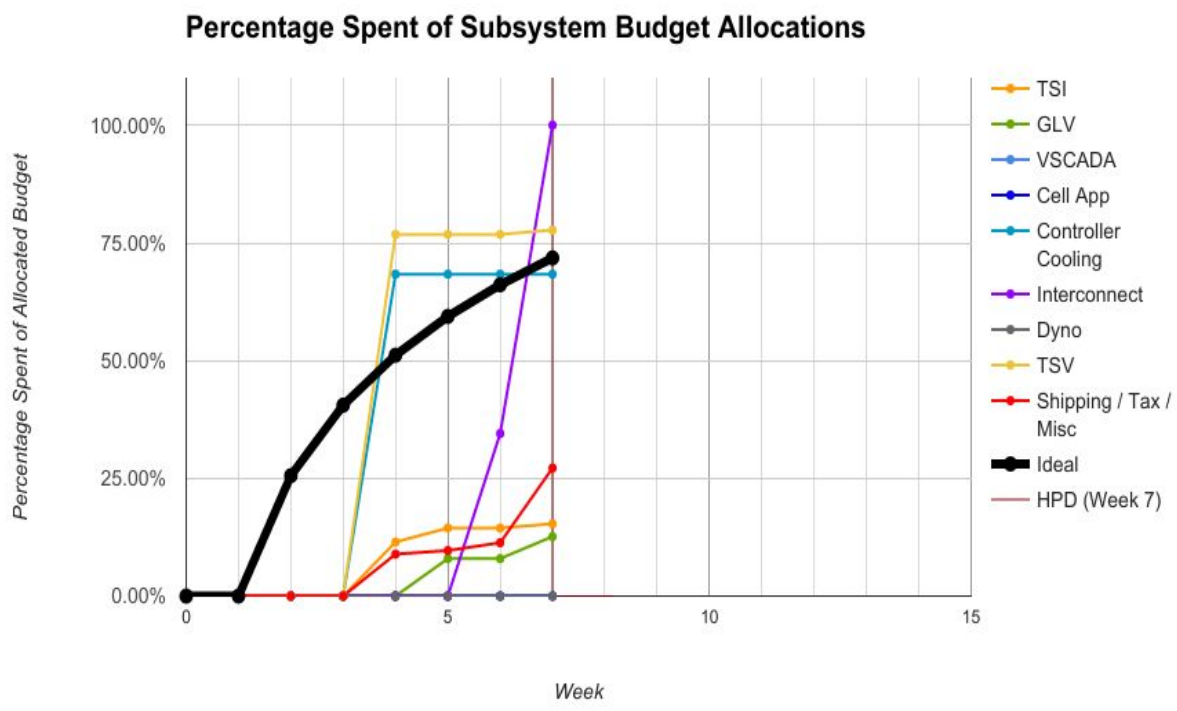
As you can tell from the table and images, we are a bit under budget from what we had originally anticipated. Part of the reason for this is because the Mechanical team has offered to pay for several of our items, such as the ~$600 GLV battery and ~$400 of subsystem boxes. Additionally, since most teams are behind schedule (see "Schedule" section below), some teams will fail to have all purchases completed by the hardware purchasing deadline. This delay means teams, such as GLV and TSI, have yet to place orders to fabricate PCBs and for all of the components they will need to populate them. We anticipate to be below budget by a reasonable amount, and purchases needed to be made in the upcoming weeks will increase.

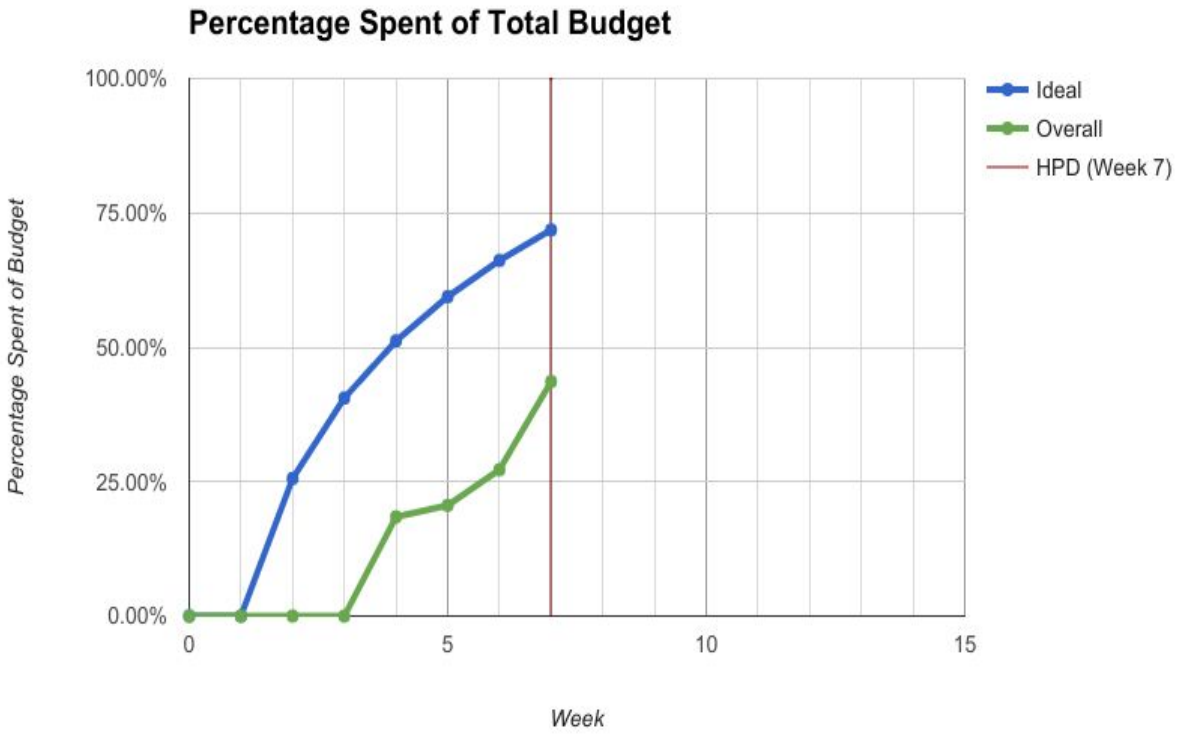
Note: All diagrams and the table below reflect purchases made until 2pm on March 8th, 2017. For the most up-to-date data and diagrams, along with all purchase orders, visit the financial section of our website.

Subsystem	Allocated Budget	Total Spent	Budget Remaining	Percentage Spent
TSI	$1,000	$153.61	$846.39	15.36%
GLV	$1,000	$126.48	$873.52	12.65%
VSCADA	$50	$0.00	$50.00	0.00%
Cell App	$125	$0.00	$125.00	0.00%
Controller Cooling	$600	$410.47	$189.53	68.41%
Interconnect	$1,000	$1000.62	-$0.62	100.06%
Dyno	$50	$0.00	$50.00	0.00%
TSV	$500	$388.95	$111.05	77.79%
Shipping / Tax / Misc	$1,175	$319.52	$855.48	27.19%
TOTAL	**$5,500**	**$2399.65**	**$3,100.35**	**43.63%**


Allocated Budget
TSI
GLV
VSCADA
Cell App
Controller Cooling
Interconnect
Dyno
TSV
Shipping / Tax / Misc
$1,000
$1,000
$600
$1,000
$500
$1,175


Budget Spending
Total Spend
Budget Remaining
$1,000.00
$750.00
$500.00
$250.00
$0.00
TSI
GLV
VSCADA
Cell App
Controller Cooling
Interconnect
Dyno
TSV
Shipping / Tax / Misc





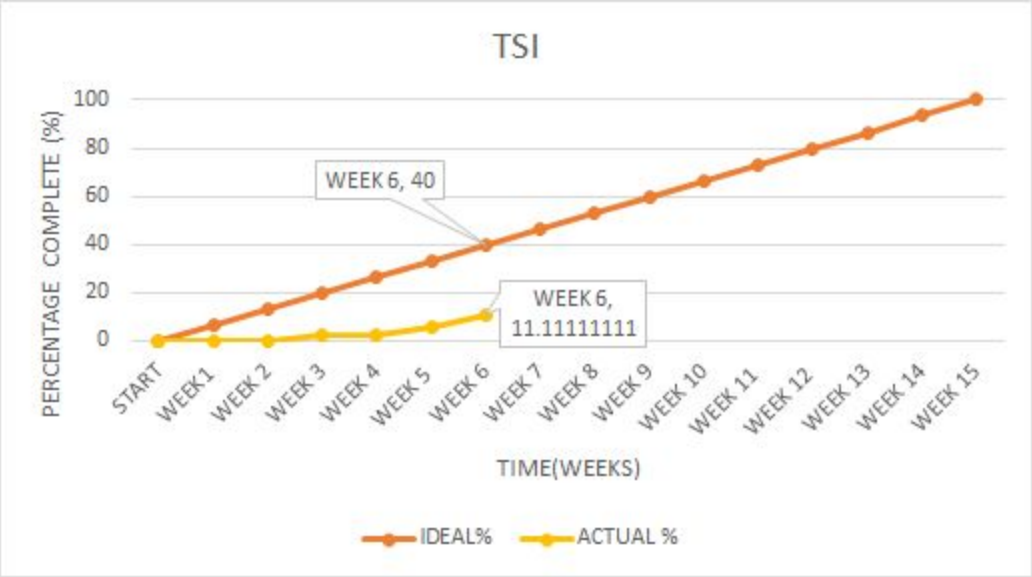
*HPD = Hardware Purchasing Deadline (March 9th, 2017)

** Ideal curves are based off of a simple exponential graph since purchasing was expected to grow exponentially across the semester.

Schedule

The schedules are done based off each individual system's WBS, which can be found on the WBS page of the website. Represented below are the graphs of the current state of affairs, based off Individual Progress Reports (IPR). These are culminated into weekly Project Status Letters (PSL), which also can be found on the website's PSL page. Our goal is to be no more than 20% off track, however, there are some unkown unkowns that we cannot anticipate. It should be noted that PSL did not start until week 4, as many teams were getting acclimated and WBSv0.1 was being structured.

TSI- Tractive Systems Interface



Causes of delay:

- There was an initial delay of 1.5-2 weeks as the engineers began to acquaint themselves with all pertinent resources and materials.
- The crux of TSI design is a PCB, which is undergoing constant and iterative review.
- ESF forms required a thorough and detailed cataloguing of all design choices, which took a week.
- Poor existing documentation required further attention.

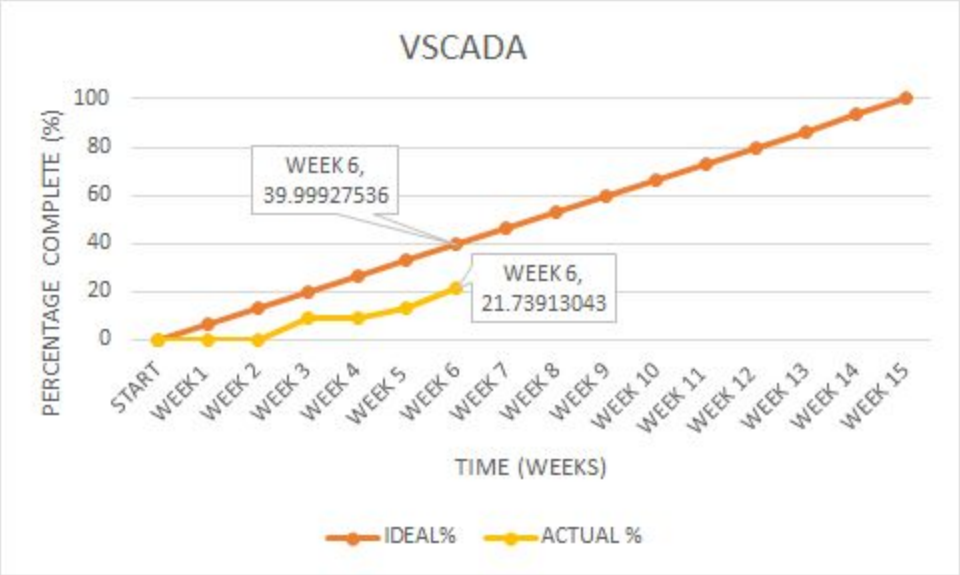
Plan to mitigate:

- Seek/ consult with engineer Flynn's expertise in PCB design layout.
- Any further competition forms will be handled by Management and other engineers.

Breakdown:

- 36 tasks at present moment x 2.778 weightage = 99.72% of total TSI system

VSCADA- Vehicle Supervisory Control and Data Acquisition



Causes of delay:

- Initial design discrepancies between MongoDB and generic DB took a week to decide best course of action.
- Existing code maxed out a core of the processor, therefore, they rewrote the code in a more robust language.
- Views layout design required extra time.
- Some design choices needed to be compatible with Cell App team.

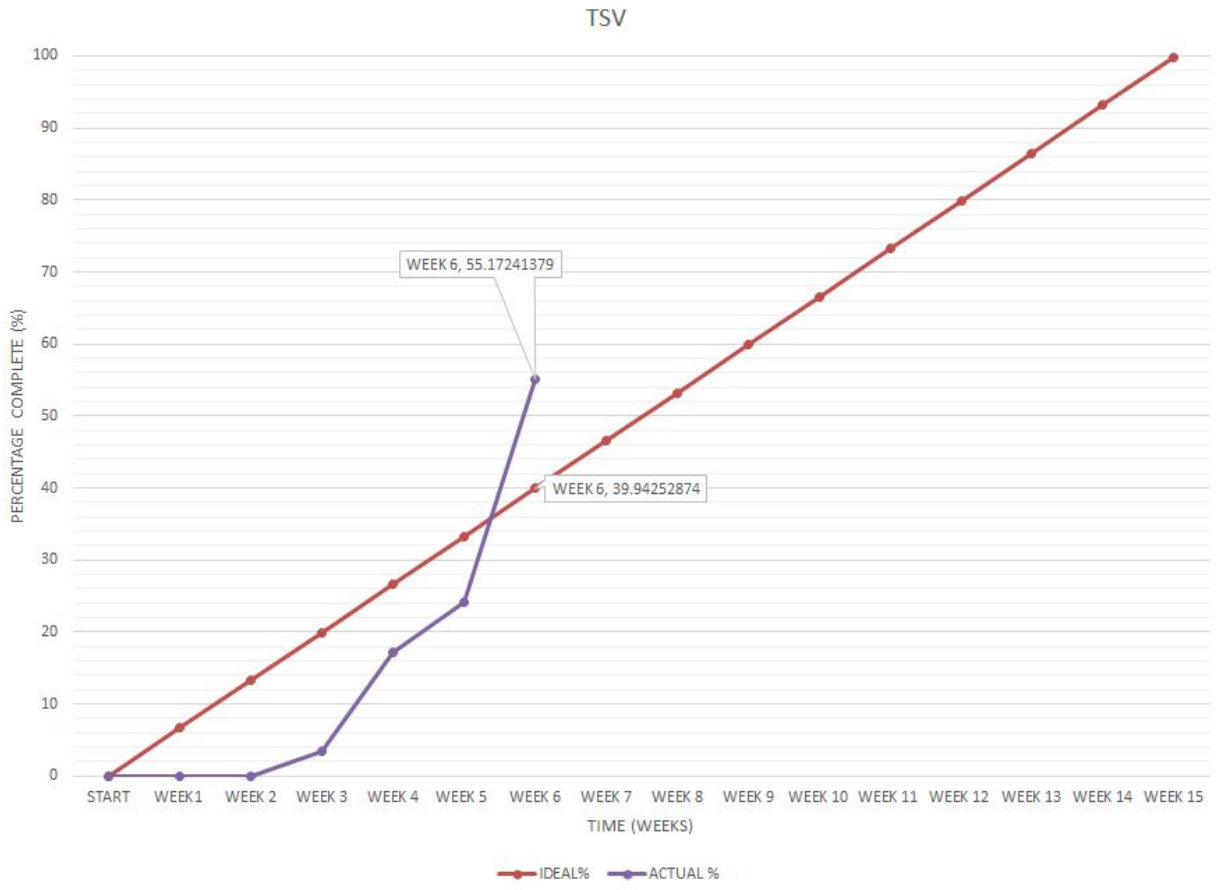
Plan to mitigate:

- Engineers Lombardo and Wiles will use Java DB with a SQL backend.
- Rewrote code from Java and Python.
- Outsourced layout generation to engineer Beggs.

Breakdown:

- 23 tasks at present moment x 4.34 weightage = 99.82% of total VSCADA system.

TSV- Tractive Systems Voltage



Causes of delay:

- There is no delay, as this team had the most usable previous years' work.
- Engineers Guro and Grybos did not have design work, and were implementing existing design.

Plans to Mitigate:

- There is no need for a mitigation plan. They are ahead of schedule.

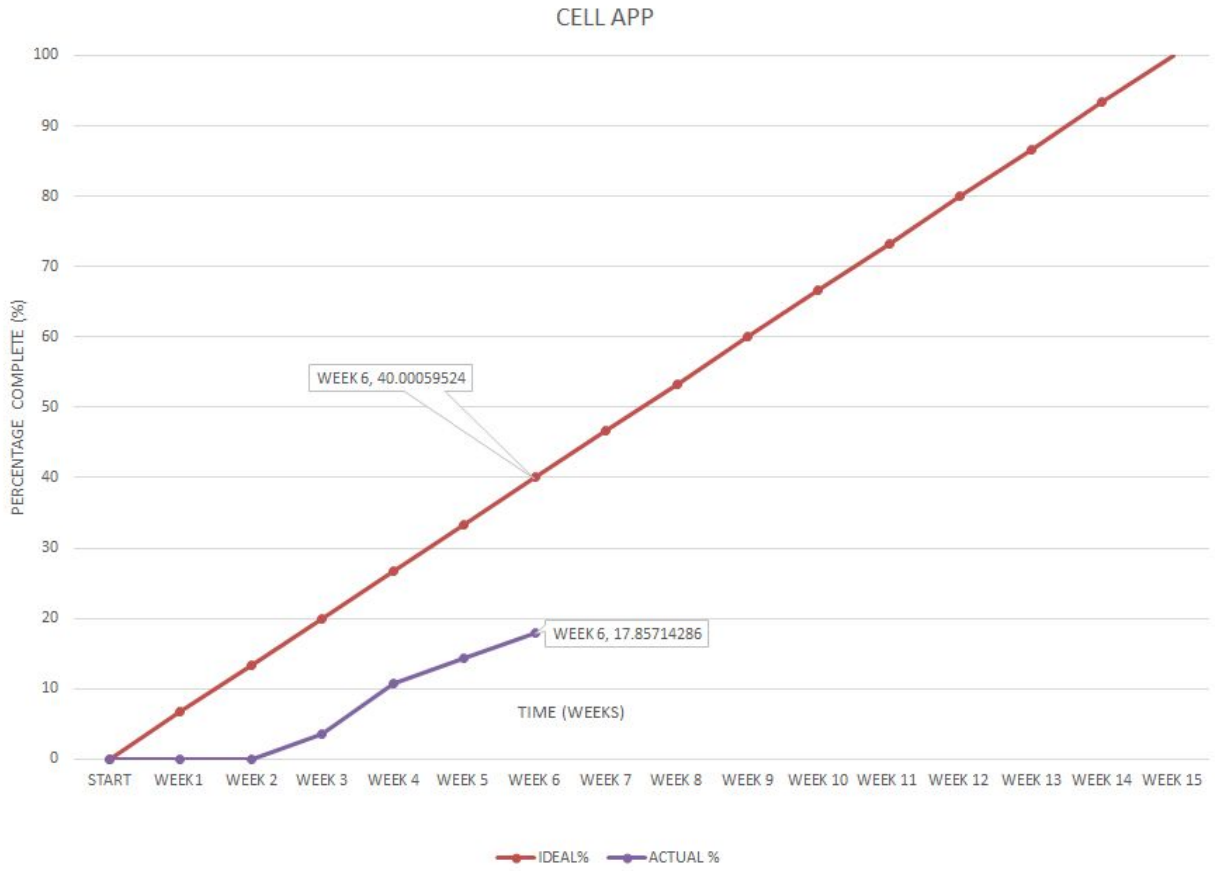
Breakdown:

- 29 tasks at present moment x 3.44 weightage = 99.76 % of total TSV system.

Further note:

- Team plans on being able to sufficiently implement basic functionality by week 8. Afterwards, they will improve upon existing design/aid other teams with their potential backlogs.

Cell App- Cell Application



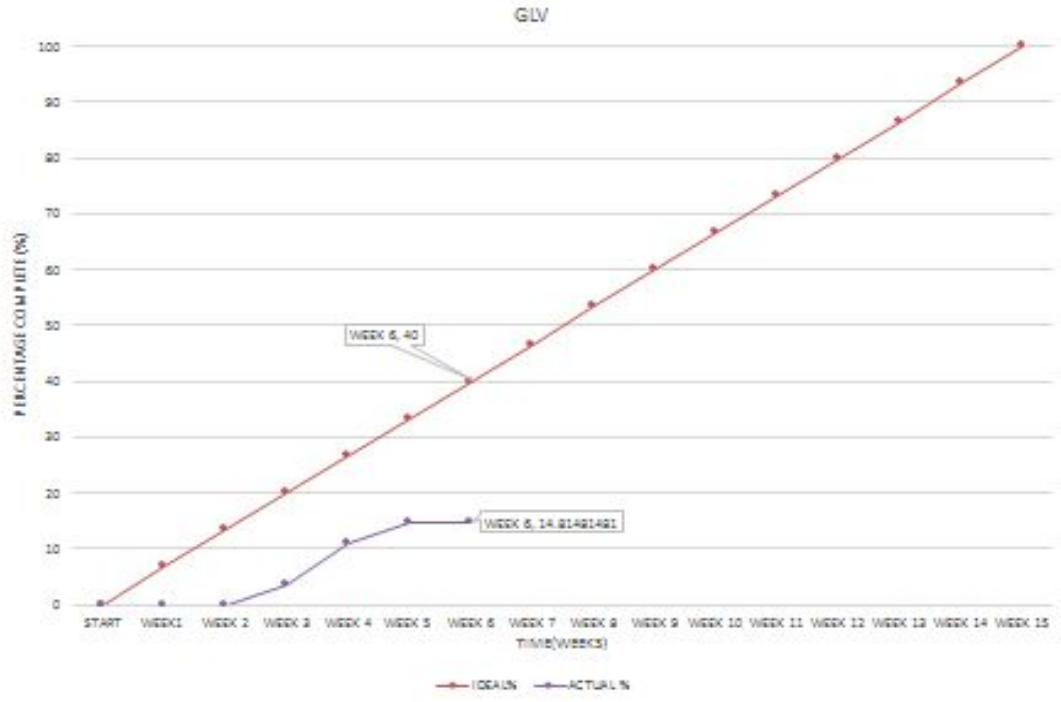
Causes of delay:

- There did not exist any cell application functionality before this year, therefore, the engineers had to start from the foundation.
- The team was working closely with the engineers on VSCADA, and had an iterative and ongoing review process. (databases, data handling etc.)
- There is an inherent dependency in that the application needs data from the VSCADA team to display.
- Engineers will use

Plan to mitigate:

- Engineers Dilsiz and Birru will be using “dummy data” with reasonable defaults to maintain feasibility, to populate displays they have created.

GLV- Grounded Low Voltage



Causes of delay:

- Initial hesitation of respinning BoB versus adding a separate board for current sensor.
- ESF forms required 1 week of potential design time.
- Hesitation of purchasing/designing box caused several days' hiccup.
- Waiting for parts to arrive prevented engineers Bennett, Phillips, and Sluke from testing to ensure functionality within a day.

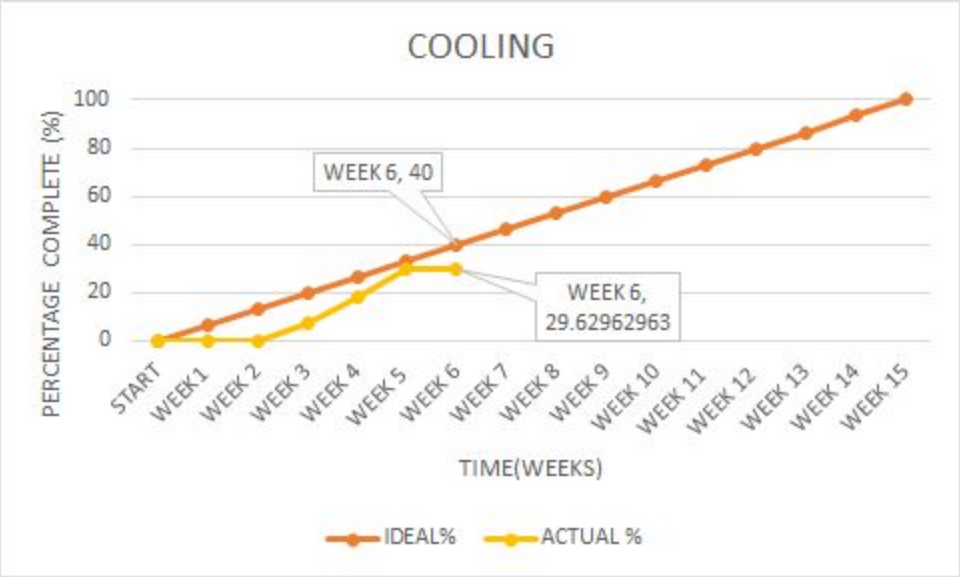
Plan of mitigation:

- Seek/consult engineers Flynn/Townley's experience in design.
- Purchase box with help of MECH-E team.

Breakdown:

- 27 tasks at present moment x 3.703 weightage = 99.9 % of total GLV subsystem.

Controller Cooling System



Causes of delay:

- ESF forms caused a work halt for about 2 days.
- Waiting for parts to arrive prevented engineers Guo and Han from testing to ensure functionality within a day.
- Electrical schematic approval is an ongoing and iterative progress.

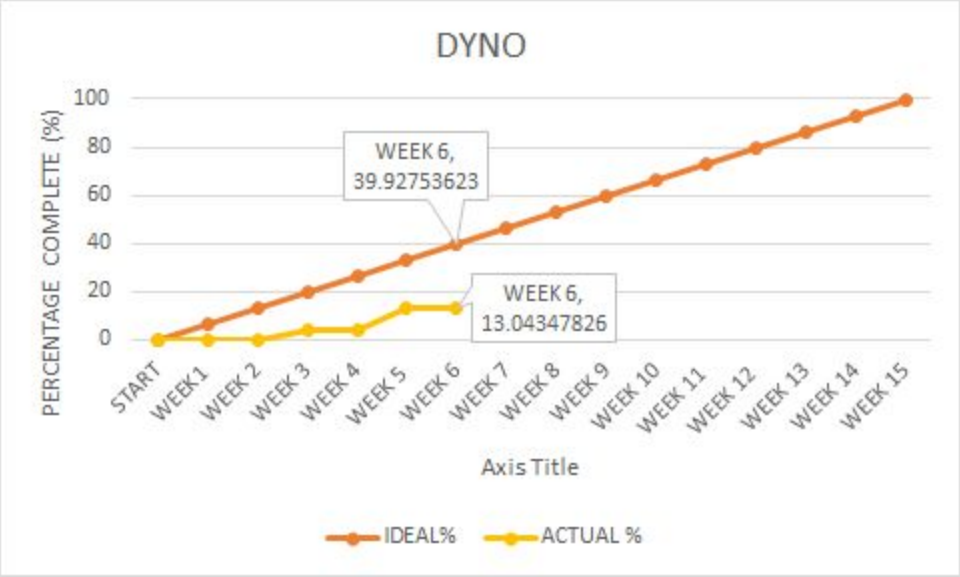
Plan to mitigate:

- Team is advised to build to specifications of commercial system or existing system in Dyno such that they can swap if necessary.

Breakdown:

- 27 tasks at the present moment x 3.703 weightage = 99.9% of cooling system

DYNO- Dynamometer



Causes of delay:

- Existing documentation was bleak and code was not commented to a usable grade.
- Engineer Diego needed to rewrite portions in Python.

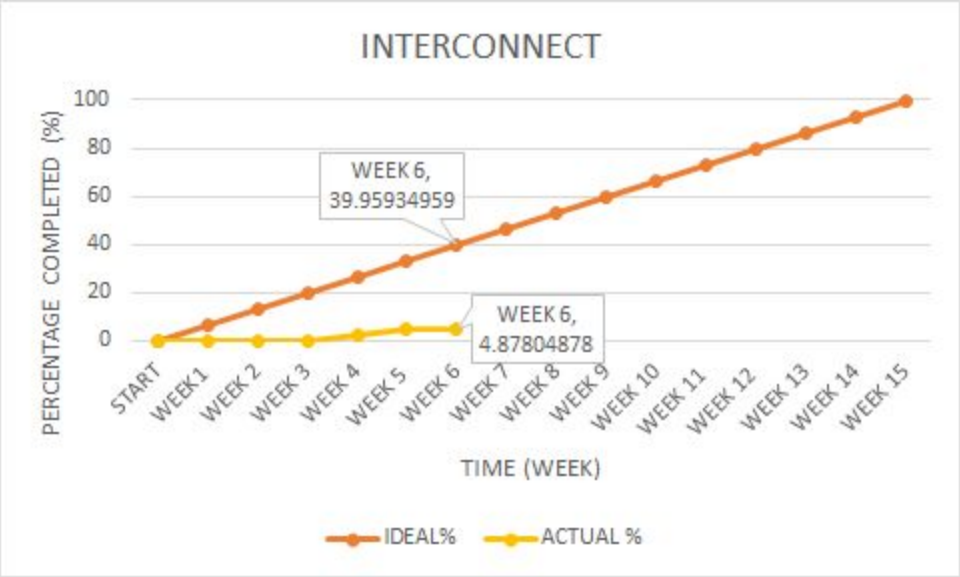
Plan to mitigate:

- Consult with engineers Martocci/Flynn for best course of action.

Breakdown:

- 23 tasks at present moment x 4.34 weightage = 99.68 % of total system

IC- Interconnect



Cause of delay:

- Waiting for parts to arrive prevented engineers Chiesa and Port from assembling them.
- The system block diagram has been revised many times, and engineers needed to keep tracks of updates/refactor inventories/update .
- Double shielded cable research/ plan was still unsure of which cable exactly to purchase

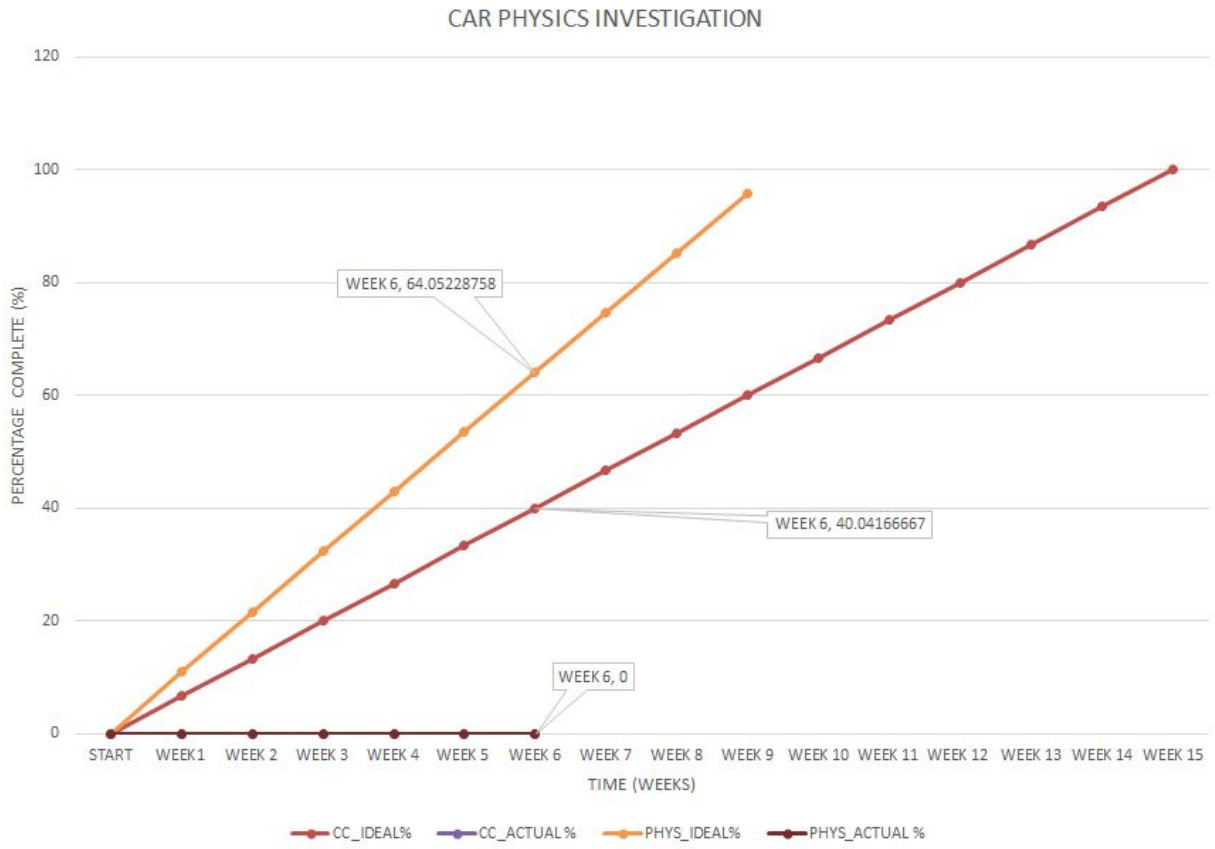
Plan to mitigate:

- Due to the large number of cables to assemble, we anticipate that any member of Management/other teams to help in this assembly.

Breakdown:

- 41 tasks at present moment x 2.43 weightage = 99.63 % of interconnect subsystem

PHYS/CC- Car Physics Investigation / Cruise Control Modeling



Cause of Delay:

PHYS:

- The data that existed was not sufficient/ was not accurate enough to draw correct conclusions.
- Iterative review process that needs constant feedback.
- Further testing needs to be implemented

CC:

- The data that existed was not sufficient/ was not accurate enough to draw correct conclusions.
- Further testing needs to be implemented

Plan to mitigate:

PHYS: Seek further guidance on memos (Prof, Boekelheide/Mech.E. professors)

CC: Seek further guidance on memos (Prof, Boekelheide/Mech.E. professors)

Breakdown:

PHYS: 17 items at the present moment x 5.88 weightage = 99.96 % of Physics modeling delivered

CC: 4 items at the present moment x 25% weightage = 100% of Cruise Control Modeling delivered

MGMT- Management

Causes of delay:

- WBS took a while as teams were getting acclimated.
- PSL were incorrect at times (items were thought to be approved when they weren't).
- ESF forms took a larger period of time than expected

Plan to mitigate:

- Proposed "further investigation" column on PSL so not to confuse unapproved items as approved.

Breakdown:

- 44 tasks at present moment x 2.27 weightage = 99.88 % of total MGMT subsystem

Appendix A: Schematics

TSV
PackMAN
AMS

Appendix B: Bill of Materials

The most up-to-date bills of materials can be found on our website:
http://sites.lafayette.edu/ece492-sp17/management/financials/bills-of-materials/

Appendix C: Safety Plan

We have not made any modifications to the safety plan. As a result we intend to use the same plan as last year.

https://sites.lafayette.edu/ece492-sp16/files/2016/03/2016SafetyPlan.pdf

Appendix D: CAN Bus ICD

https://sites.lafayette.edu/ece492-sp17/files/2017/01/CAN-Bus-ICD.pdf