Oregon State University
College of Engineering

CS Capstone Progress Report

February 16, 2018

Axolotl

Prepared for

Kevin McGrath _____ _____
Signature *Date*

Prepared by

Group 15
Team Wombat

Victor Li _____ _____
Signature *Date*

Ryan Crane _____ _____
Signature *Date*

Nicholas Wong _____ _____
Signature *Date*

Abstract

This document is a term progress report that examines the progress of the Axolotl infotainment project through Week 6 of the Winter 2018 term. The progress report will review the Axolotl project and its goals, offer a detailed account of our current progress in accomplishing project goals and implementing project components, as well as provide an overview of any challenges or problems we faced, in addition to our solutions to these problems. We will also examine the work that is left to do before the project can be considered complete.

CONTENTS

1 Group Intro

The Axolotl project has not changed from its initial vision. Our goal is still to develop an infotainment and black box system on an NVIDIA Jetson TX2 computer module. The end system still aims to combine the infotainment features expected in modern vehicles with a highly capable data logging system that is nearly non-existent in modern cars. The requirements of our project have not changed; however, some of the tools, resources, and parts utilized in the implementation of our project have deviated from those we initially specified. In this document, we will provide a recap of our project's intentions in detail whilst also discussing our own individual work with regards to our responsibilities. Each of us will offer a progress update on the work we have done, aided by code and visuals. We will also describe the work that remains to be done as well as explain some of the problems we have experienced during this period of development. Any problems we have described will be accompanied by an explanation of how we overcame those problems and/or an outline of how we plan to solve them in the future.

2 Project Overview

At its core, the Axolotl is a car infotainment system with a built in black box that logs data from various sensors in a car. The system should support the standard media playback and navigation functionalities of infotainment systems. The black box will log data captured from vehicle sensors over OBDll, footage from a connected dashcam, and data from a connected attitude heading and reference system.

2.1 Navigation

The purpose of the Axolotl navigation system is to provide users a level of navigation functionality similar to that offered by factory navigation systems installed in modern automobiles. The system should allow users to interact with a map, plan a route, as well as enter an address and navigate to the destination aided by turn-by-turn directions. The navigation system should be able to function without data service and without a GPS signal, allowing it to operate in remote locations where access to mobile data service and GPS may be disrupted. The navigation system is not designed to operate on a global scale. Instead, the system is intended to only support U.S. addresses. The navigation system should function as a subprogram of the Axolotl graphical user interface, and continue to issue route commands even if a different task is being conducted on the Axolotl system, such as management of media playback.

2.2 Media

The purpose of the media system is to give users the ability to manage and play media over their car's audio system. The Axolotl will allow users to play media from many different sources; sources that users may select from include a smartphone or media device connected through USB, Bluetooth, or Auxiliary, FM radio, media from an external USB drive, and/or internal storage. The Axolotl also allows users to set up an network connection with a remote computer and download media from that computer directly into internal storage. The media system is not intended to support all media sources; rather, it is designed to accept widespread modern media sources, inline with that offered in modern vehicles. For example, the system is not intended to support compact disc, AM radio, or tape media sources.

2.3 Data Logging

The purpose of the data logging system is to provide a level of user-friendly data logging functionality similar to that of a black box or event data recorder. The system is intended to take snapshots of vehicle information as well as video from any connected dashcams and collect that data in an easy-to-understand data log that can be easily read and interpreted from a computer. The data logging system is not intended to log the value of every variable associated with the running of a vehicle. The data fields chosen for logging are pertinent to the act of driving as well as driver well-being, however technical data fields such as those intended for mechanics are considered irrelevant and are omitted from the data log.

2.4 Video Streaming

The purpose of the video streaming or camera system is to give users the ability to utilize a dashcam in order to log vehicle data as well as provide access to live footage from a backup camera in order to improve user safety while reversing a vehicle. Thus, the video streaming system will support one dashboard camera and one backup camera. The feed from a backup camera will be streamed wirelessly to the main unit for display on the screen; this will be done so that installation will not require running long wires through the car. The backup camera will be connected to a secondary controller which will activate the camera and transmit data when the time is appropriate.

2.5 UI

The Axolotl will be connected to a touchscreen and must have a graphical user interface. The reason why Axolotl must offer a user interface is because the system needs a way to encapsulate each of its functions and allow users to manage subsystems. The user interface will enable users to interact with the system to control media playback, navigation, and access to the data log. We will be using open source Qt to develop our user interface.

3 Progress Summary by Group Member

3.1 Nick

The main goals that I had from Fall consisted of: Media, Storage Medium, and Head Unit Module. The later two were goals that were in the research documents, but once we chose one solution, there wasn't much need for development. So two more goals were added to my portion: supplemental turn signals, and networked media. The media portion was broken down into 4 smaller sections: The player software itself, bluetooth device streaming, auxiliary connectivity, and FM radio.

3.1.1 Work Completed

When researching a media player in the design document, it was decided that Kodi media player would be the final player, but once I started developing, I encountered a few errors with installing Kodi media player with dependencies. The specific errors that occurred during the installation are:

```
The following packages have unmet dependencies:
 kodi : Depends: kodi-bin (>= 2:17.6+git20171114.2125-final-0xenial) but 15.2+dfsg1-3ubuntu1.1 is
        Depends: libshairplay0 but it is not installable
        Depends: libcec4 but it is not installable
        Recommends: libvdpau1 but it is not going to be installed
        Recommends: i965-va-driver but it is not installable or
                    libva-intel-vaapi-driver but it is not installable
E: Unable to correct problems, you have held broken packages.
```

Thus, I decided to install VLC media player as a backup until a solution was made. Initial progress for receiving data from an FM antenna through I2C has been made. This code supplied by the manufacturer however only works with a specific Arduino board where specific hardware layout is defined. In order to allow the code from the Arduino sketch to function properly, it needs to be refactored to fit the Jetson TX2's hardware layout. The current problem that is impeding this goal is that when compiling the current code to establish a connection with the antenna, the built in functions from the linux library i2c-dev.h present an error:

```
error: use of undeclared identifier 'i2c_smbus_read_byte'
```

Another problem that will have to be addressed in the future is that once the data from the antenna is received, the Jetson must know how to handle it. A solution for this problem has not yet be devised but is currently being researched.

Auxiliary connectivity has not yet been implemented yet, as the hardware still needs to be ordered. Network media also has not been implemented yet, however research has been done into the the connection methodology. A connection through TCP/IP will be used to link the Axolotl to a home music library over the internet, and a program will transfer any music files present on the home music library to the Jetson.

With regards to supplemental turn signals: this goal included connecting turn signals to the Jetson wirelessly. A solution was made for the Jetson TX2 to be connected to a RaspberryPi through bluetooth and send it signals for turning on an LED. The first step of this goal was to setup the RaspberryPi as a receiver, and then the Jetson TX2 as the sender. This was done through a C program found in "An Introduction to Bluetooth Programming" book. Ultimately, the RaspberryPi was successfully paired with the Jetson and can receive signals from the Jetson wirelessly. The hardware for

the turn signals have yet to be picked up, but the RaspberryPi should be able to simply turn on the LEDs upon receiving a specific signal. A problem with this goal was that Bluetooth only allows for 1 connection for transmitting/receiving. This means that if the media player is streaming music through bluetooth, and the user wishes to send a signal to change music while sending a turn signal to the supplemental turn signals, there will be an error. A solution to this is it will be defined that the user can only send one bluetooth signal at a time in the final product specifications.

3.1.2 Planned Work

In order to allow the code from the Arduino sketch to function properly, it needs to be refactored to fit the Jetson TX2's hardware layout. Another problem that will have to be addressed in the future is that once the data from the antenna is received, the Jetson must know how to handle it. A solution for this problem has not yet be devised but is currently being researched.

For bluetooth streaming to work, the command pacmd, and the loading of a module need to be executed in the background. Thus for this goal, a script still needs to be written to automate the process of pairing a phone with the Jetson TX2.

3.1.3 Problems Encountered & Solutions

A major error that occurred while developing most of the goals was with the

```
sudo apt-get install
```

command. When running this command, the Jetson would output an error:

```
E: Could not get lock /var/lib/dpkg/lock/ - open (11: Resource temporarily unavailable).
```

In order to work around this error, the specific lock folder was deleted with the

```
rm -rf
```

command. The implications behind this removal are still unknown, but there are still no known errors after the removal.

With regards to Bluetooth Streaming: an iPhone was successfully paired with the Jetson TX2 through bluetooth, and was able to output audio through the speakers connected to the Jetson (Hdmi). This was done by following the source: https://blog.stevenocchipinti.com/2012/10/bluetooth-audio-streaming-from-phone-to.html/. However a few changes were necessary as some commands were outdated. First, the following line was added to /etc/bluetooth/main.conf (instead of audio.conf):

```
Enable=Gateway,Source,Socket
```

Then instead of using d-feet to connect to a device, bluetoothctl was used. Then the following line was added to /etc/pulse/daemon.conf, after the line "resample-method = speex-float-1":

```
"resample-method=trivial"
```

There was a problem that occurred when trying to connect a device to the Jetson where attempting to connect would result in the error:

```
Failed to connect: org.bluez.Error.Failed.
```

This error was fixed by running the command:

```
pactl load-module module-bluetooth-discover
```

and then running the command:

```
sudo service bluetooth restart
```

After making sure that bluetooth was once again enabled, running the connect command with the desired device in bluetoothctl ended in a success. The rest of the source worked perfectly for outputting audio from device to the Jetson.

3.1.4 Code

3.1.4.1 server.c:

```
int main(int argc, char **argv)
{
    struct sockaddr_rc loc_addr = { 0 }, rem_addr = { 0 };
    char buf[1024] = { 0 };
    int s, client, bytes_read;
    socklen_t opt = sizeof(rem_addr);

    // allocate socket
    s = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

    // bind socket to port 1 of the first available
    // local bluetooth adapter
    loc_addr.rc_family = AF_BLUETOOTH;
    loc_addr.rc_bdaddr = *BDADDR_ANY;
    loc_addr.rc_channel = (uint8_t) 1;
    bind(s, (struct sockaddr *)&loc_addr, sizeof(loc_addr));

    // put socket into listening mode
    listen(s, 1);

    // accept one connection
    client = accept(s, (struct sockaddr *)&rem_addr, &opt);

    ba2str( &rem_addr.rc_bdaddr, buf );
    fprintf(stderr, "accepted connection from %s\n", buf);
    memset(buf, 0, sizeof(buf));

    // read data from the client
```

```
    while(true){    //continue running until error
        bytes_read = read(client, buf, sizeof(buf));
        if( bytes_read > 0 ) {
            printf("received [%s]\n", buf);

            //determine whether it was left or right turn
            if(buf[0] == 'l'){
                printf("LEFT SIGNAL...\n");
            }
            else if(buf[0] == 'r'){
                printf("RIGHT SIGNAL...\n");
            }
            //otherwise throw error
        }
    }

    // close connection
    close(client);
    close(s);
    return 0;
}
```

The above code runs a server process on the Raspberry Pi, allowing it to accept commands from the Jetson.

3.1.4.2 client.c:

```
int main(int argc, char **argv)
{
    struct sockaddr_rc addr = { 0 };
    int s, status;
    char dest[18] = "B8:27:EB:D8:59:7D";    //Bluetooth Address of device to connect to
    char input; //reading input from keyboard
    char buf[64];

    // allocate a socket
    s = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);

    // set the connection parameters (who to connect to)
    addr.rc_family = AF_BLUETOOTH;
    addr.rc_channel = (uint8_t) 1;
    str2ba( dest, &addr.rc_bdaddr );
```

```
    // connect to server
    status = connect(s, (struct sockaddr *)&addr, sizeof(addr));

    // send a message
    if( status == 0 ) {
        while(true){        //persistent data transfer
            do{
                scanf("%c", &input);        //read from keyboard input
            } while(input != 'l' | input != 'r');
            buf[0] = input;
            status = write(s, buf, 1);    //send 1 char to server
        }
    }

    if( status < 0 ) perror("error in sending data");

    close(s);
    return 0;
}
```

The above code runs a client process on the Jetson, allowing commands to be sent to the Raspberry Pi controlling the turn signals.

3.1.5 Visuals

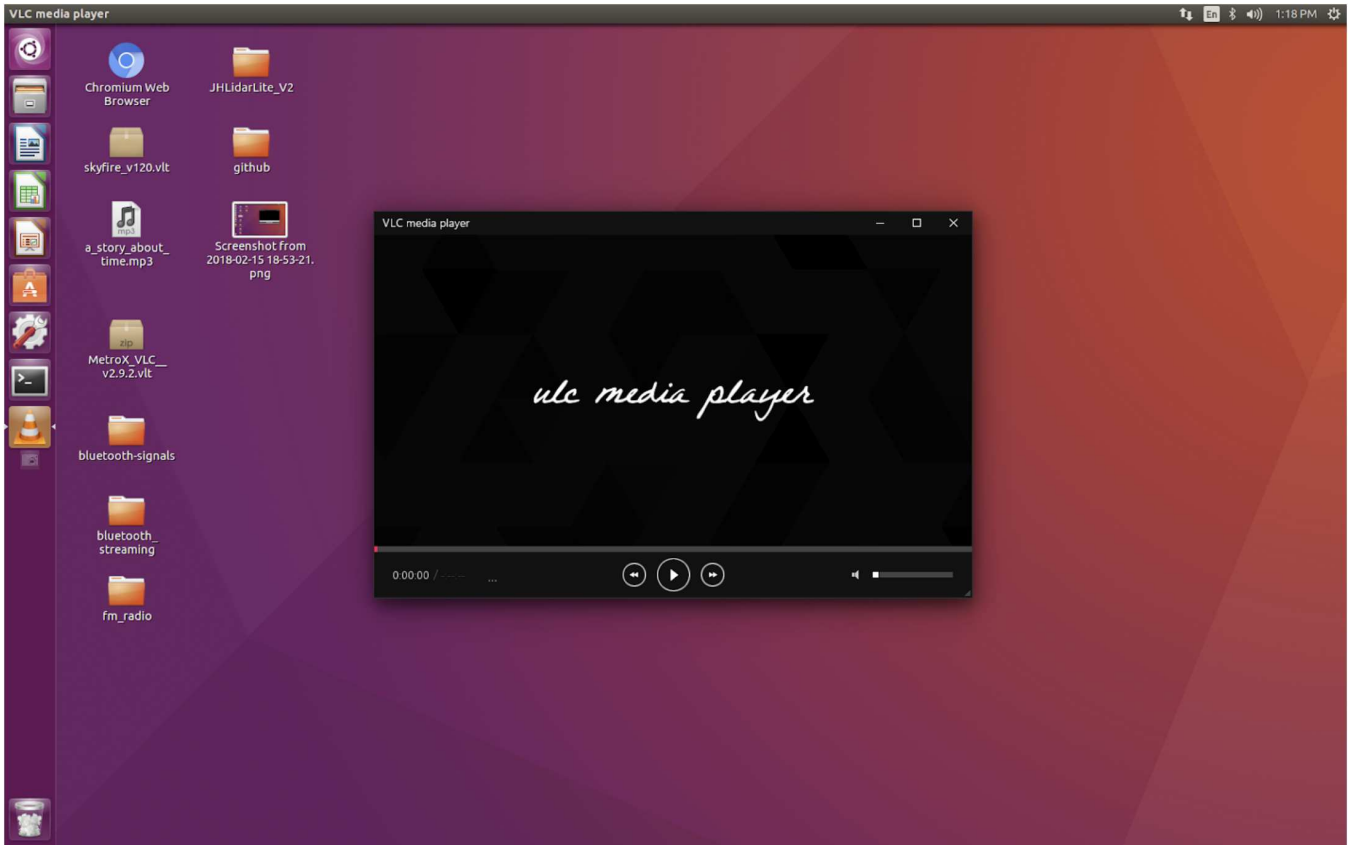

Fig 1.1: VLC media player working on the Jetson.

3.2 Ryan

The goal for the project is broadly to develop an infotainment system which integrates data logging and wireless backup camera streaming. My individual development goals for this term are the user interface and the wireless video stream. Thus far I have mostly been focused on making the interface, and more recently integrating some of my partners' work into it.

3.2.1 Work Completed

3.2.1.1 User Interface: We decided to use Open Source Qt as a framework to develop the user interface, which is a huge set of tools that I had no useful experience experience with prior to this project. Much of the first two weeks of work was focused on trying to set up Qt's IDE and frustrated searching for learning resources online. Eventually I was able to put together a startup screen, main menu page and a music page. The main menu has three buttons corresponding to music, navigation, and the black box and data portions of the project. The music page has functionless buttons for the various sources we will be supporting and a back button to return to the main menu. I have also written a class in C++ and registered it as a QML type in order to fork and exec Victor's data logging processes to run in the background. This runs from the startup screen as it transitions to the main menu. Integrating the navigation application will be a little more complicated since we need to be able to move between navigation and other pages, rather than simply launching it in the background forever.

3.2.1.2 Video Streaming: We decided to use a Raspberry Pi with a Raspberry Pi Camera module as the backup camera. I have setup the operating system, configured the camera and installed some important software packages. Generally the interface development has been a higher priority so far.

3.2.2 Planned Work

3.2.2.1 User Interface: At this point we have a usable base interface to for integration with the backend features that are in development. Another QML page will probably need to made to access the data log for download and deletion. Most of the required work ahead for the interface is just connecting it to the implementations of its features. If time allows I would like to try and make the user interface prettier, since it is quite bland at the moment. Integrating the navigation application is the most immediate task. Integrating the data logging programs allowed for a simple solution to fork a child process and exec the program since it runs in the background unseen. Navit would have to be closed and executed again to transition pages under this solution which wouldn't feel much like it was part of the project, so a more intricate solution will be required.

3.2.2.2 Video Streaming: I have focused most of my efforts on the interface at present. There is an open source tool called mjpg-streamer that I believe will work for our purposes. Once streaming the video from the Raspberry Pi to the Jetson is set up we want to wire the Jetson to the car's reverse lights to detect when the car is in reverse and switch the video on the monitor over to the camera, and back when the car leaves reverse. All of this still needs to be done.

3.2.3 Problems Encountered & Solutions

The largest impediment to my progress has been the learning curve for qtcreator and the various tools accessible through Qt. It took me about a week to get compilers configured in qtcreator just to run the sample projects that it came with. Since I've found my way around the IDE and learned enough basic QML to make our interface pages everything has progressed about as smoothly as it can have gone on that front. Curiously when I was setting up the Raspberry Pi and

installing packages it started looping the login screen indefinitely after a reboot. I had no idea what caused that and just pushed a fresh os install to make it stop. That wasn't a terrible inconvenience but it was quite bizarre.

3.2.4 Code

3.2.4.1 QObject:

```
Class test : public QObject{
Q_OBJECT
Public:
Explicit test (QObject* parent = 0) : QObject(parent){}
Q_INVOKABLE int start_navit();
Q_INVOKABLE int start_dm();
};
```

This is the header for a class I wrote to launch the other processes we will need to fulfill the requirements of our project. Most of it is simple C++ and the rest is some wizard magic that helps Qt work with it. There is another line of code shown below which creates a QML type out of it and allows us to use it's functions in the interface:

```
qmlRegisterType<test>("forker.us",1,0,"Test");
```

The string arguments to this call are the names of the package that must be imported to us my type and the name of the type in QML. The numeric arguments make the version number of the package, in this case 1.0.

```
MainMenu { id: pagetwo }
MusicPage { id: pagethree }
StackView {
Id: stack
initialItem: startup
}...
stack.push(pagetwo)
```

This code transitions pages in the user interface. The first two lines assign names to qml types that I defined in other files and then a stackview object is declared which provides a stack-like interface for performing screen changes. In addition to push and pop stackview also has a replace function tah can replace the item currently at the top of the stack with another page. This could be used for lateral transitions between pages at lower depths without being forced to return to the menu first.

3.3 Victor

My responsibilities this term were to implement the navigation and data logging system for our project, as well as implement low-level functionality such as the Axolotl's wake and sleep logic based on vehicle start and vehicle shutoff.

3.3.1 Work Completed

Navigation has been implemented using Navit navigation software, as specified in our design document. Navit has been installed and fully configured. The NEO M8U GPS chip has been integrated into the navigation system with some modifications to the original USB 1.0 connection method, aided with the addition of a supplementary device driver. As of this point in time, the navigation system's core functionality is present. The system is able to fully function as a navigation system without the need of an internet connection. This offline capability includes the ability to determine the user's current location, the ability for users to enter their desired destination, and the ability for the system to calculate a route from the user's current location to their destination, producing both a route itinerary as well as navigation instructions. The system can also recognize changes in the user's position and update the route and navigation instructions accordingly, whether through the GPS chip or dead reckoning system, and can also handle route cancellation. Minor UI modifications were made in order to make the map view of the navigation system easier to understand and use.

The data logging system has been implemented in C++, utilizing multiprocessing. The system is subdivided into three subsystems: the OBDII logging system, the AHRS logging system, and the dashcam logging system. One daemon to handles OBDII and AHRS logging and the other handles dashcam logging. A daemon manager controls and manages these daemons, and Python adapter script supports all interactions with the OBDII port. All supporting file system functions have also been implemented in a .hpp header file that is imported by the daemon manager and the two daemons.

All of the aforementioned components work as intended. The daemon manager works sets up the logging environment and then spawns two daemons. The data logging daemon contains an interface to log the AHRS data and loops the execution of the Python adapter script to record OBDII data. The dashcam daemon is effectively a clone of the data logging daemon, with an interface prepared to access dashcam footage. The Python adapter script implementation also has functions to handle functionality to handle fetching and clearing DTC codes from the OBDII port, which only need to be called from the UI. Additionally, all helper functions for passworded access to the data logging system's delete functionality have been implemented.

With respect to individual data logging subsystems, OBDII logging is functionally complete. AHRS and dashcam logging is not currently functional, however interfaces for these features are present to allow for rapid integration of those features at a later date.

The OBDII data logging system has been functionally tested in real driving environments with good results. Functional testing has revealed that the system logs data competently once every 19 to 21 seconds. Performance was very stable from the test, with the system only failing to retrieve OBDII data during engine start and stop, understandable due to the power flux of the OBDII port during that time. All of these test results were obtained from a 50-minute driving loop around Corvallis at mostly city speeds of 0 to 50 kph (35 mph).

3.3.2 Planned Work

The navigation system is functional at this point in time, however additional modifications are needed. An address entry option needs to be added to the destination entry menu as the system is currently unable to accept a US address and translate it into a destination, a key requirement outlined in last term's software requirements specification. Additionally, UI improvements need to be made to the navigation GUI as the system's default UI is unintuitive, hard to interpret, and somewhat dangerous to utilize on the road. Both tasks can easily be accomplished by rewriting Navit's XML configuration file.

The OBDII data logging system is fully functional at this point in time, though efficiency enhancements are need to be made. Currently, the system requires 19 to 21 seconds to log 11 fields of data to the .csv log file. This is far too slow to support our fuel efficiency monitor stretch goal, which requires near real-time data to be streamed from the OBDII port. The python adapter script for OBDII needs to be rewritten to use non-blocking OBDII queries to reduce the time cost of each call to the OBDII port. A custom query to fetch all 11 fields in one bytestream may also need to be written if the non-blocking queries do not achieve the performance desired. AHRS integration also needs to occur as the data logging daemon in its current form does not log AHRS data. The data logging daemon already has an interface ready to accept the additional logic necessary to log AHRS data to the .csv log file, thus the only task left is to acquire the AHRS chip and integrate it into the system. Once AHRS integration is complete, additional software can be written to achieve our live AHRS data stretch goal. Finally, the recording functionality of the dashcam daemon needs to be implemented. Our video streaming pipeline is incomplete, thus the dashcam daemon's record() function does not yet do anything. Once a video pipeline is established to the Raspberry Pi cameras, a small addition to the dashcam daemon's record() function is all that is needed to complete the dashcam logging system's implementation.

Custom wake and sleep logic needs to be implemented in order for our system to meet spec. Specifically, the Axolotl must turn on when the vehicle is turned on, and enter a low power mode for 48 hours when the vehicle turns off. This will need to operate independently from the TX2's power button. A switch will thus wired into a GPIO pin to assert system on and system off. A program will then be written to monitor the GPIO pin's value and execute the appropriate sleep or wake functionality on state change. Research into wake-enabled pins as well as the process of entering low power mode on the TX2 has been completed.

3.3.3 Problems Encountered & Solutions

The main issue that impeded my progress involved hooking up the NEO M8U GPS receiver to the navigation system. Originally, a step down from USB 3.0 to USB 1.0 was required, necessitating the use of a GE USB adapter. However, this adapter significantly reduced the reliability of the GPS chip; a light bump would be enough to unseat the adapter's power pins and cut power to the chip. The Jetson was also unable to recognize the GPS chip when the adapter was connected through a USB 3.0 hub (necessary for implementation and testing). To bypass this, I connected the GPS chip directly to the USB hub, allowing the Jetson to recognize the GPS chip as a serial device. I then installed an additional driver called cdc_acm which allowed the GPS receiver to be enumerated and be used by Linux's gpsd (GPS daemon) without the need to step down to USB 1.0.

Another problem that impeded my progress was the fact that our original python-based OBDII logging library failed to support a number of the PIDs (data fields) that we wanted to log. I hoped to compensate for the unsupported PIDs by selecting different PIDs that were supported. Our client wanted to stick as close to the previously approved spec as possible, so we elected to use a different OBDII logging library that supported all of the functionality we needed.

In terms of pressing problems, there are few. The most significant is the OBDII data logging system's low sample rate. Functional testing has revealed that the OBDII logging script takes 19-21 seconds to sample the OBDII port and log 11 fields. This is much slower than our desired sampling speed of 1 sample per second, thus making it far too slow to support our fuel economy stretch goal. As explained in the work remaining section, a rewrite of the python-OBD adapter script to use asynchronous non-blocking OBDII queries and/or a custom OBDII query to fetch the desired OBDII data in a single bytestream should net the performance improvements needed in order to reduce the time cost of each set of OBDII queries.

3.3.4 Code

3.3.4.1 Data Logging Directory Builder:

```
string buildSaveDirectory() {
  int buildStatus = 2;
  string dirName = axolotlFileSystem::getHomeDir() + "/axolotl/data/axolotl_log_";

  time_t startTime = time(NULL);
  struct tm *startTimeAsUTC = gmtime(&startTime);
  string startYear = to_string(startTimeAsUTC->tm_year+1900);
  string startMonth = to_string(startTimeAsUTC->tm_mon+1);
  string startDay = to_string(startTimeAsUTC->tm_mday);

  int startSecProcessed = startTimeAsUTC->tm_sec;
  string startSec = to_string(startSecProcessed);
  if(startSecProcessed < 10) {
    startSec = "0" + startSec;    // ensures that seconds are always 2 digits
  }

  int startMinProcessed = startTimeAsUTC->tm_min;
  string startMin = to_string(startMinProcessed);
  if(startMinProcessed < 10) {
    startMin = "0" + startMin;    // ensures that minutes are always 2 digits
  }

  long startHourProcessed = startTimeAsUTC->tm_hour+PST;
  if(startHourProcessed < 0) {
    startHourProcessed += 24;   // adjusts if UTC-8 is earlier than UTC midnight
  }
  string startHour = "" + to_string(startHourProcessed);
  if(startHourProcessed < 10) {
    startHour = "0" + startHour;    // ensures that hours are always 2 digits
```

```
  }
  dirName = dirName + startYear + "_" + startMonth + "_" + startDay + "_" + startHour + startMin +
  buildStatus = mkdir((const char *)dirName.c_str(), S_IRWXU | S_IRWXG | S_IRWXO);

  if(buildStatus != 0) {
    dirName = "__fail_dir_build";
  }
  return dirName;
}
```

The above code exists in my daemon manager, which manages the data logging and dashcam logging daemons. This code is run before either daemon is executed and is used to build the directory used to hold the data logged during the current boot cycle. Every boot cycle produces a new logging directory; every logging directory is named based on the date and time of when it was created.

3.3.4.2 axolotlFileSystem class:

```
class axolotlFileSystem {
  public:
    static double getAvailableMemory(std::string volumePath) {
      boost::filesystem::space_info volStats = boost::filesystem::space(volumePath);
      return (double)volStats.available/1048576;
    }

    static std::string getHomeDir() {
      struct passwd *userPath = getpwuid(getuid());
      return userPath->pw_dir;
    }

    static std::string getPWD() {
      char pwd[2048];
      getcwd(pwd,sizeof(pwd));
      return (std::string)pwd;
    }

    static std::string hash(std::string s) {
      // Salt and hash
      std::string hashedPass, passSalt = "thequickbrownfoxjumpsoverthelazydog";
      CryptoPP::SHA256 passwordHash;
      BYTE passwordDigest[CryptoPP::SHA256::DIGESTSIZE];
      passwordHash.CalculateDigest(passwordDigest,(const BYTE *)passSalt.c_str(),passSalt.size());
```

```
        // Encode into human-readable string via hex
        CryptoPP::HexEncoder hashEncoder;
        CryptoPP::StringSink *hashStringSink = new CryptoPP::StringSink(hashedPass);
        hashEncoder.Attach(hashStringSink);
        hashEncoder.Put(passwordDigest,sizeof(passwordDigest));
        hashEncoder.MessageEnd();

        // Return the string hash
        return hashedPass;
    }
};
```

The above code defines a class with static functions accessible by prepending the function name with ”axolotlFileSystem::”. These functions are accessible by the daemon manager as well as the data logging and dashcam managers, and are used to handle the data logging system’s numerous interactions with the TX2’s file system.

3.3.5 Visuals

@@Thu Feb 8 18:57:31 2018	27.0588235294 percent	604.25 revolutions_per_minute	0 kph	13.7254901961 percent	2.35294117647 percent	1543 second	22.7450980392 percent	88 degC	None	14 degC	101 kilopascal
@@Thu Feb 8 18:57:52 2018	34.1176470588 percent	598.25 revolutions_per_minute	0 kph	14.5098039216 percent	2.74509803922 percent	1564 second	22.7450980392 percent	88 degC	None	14 degC	101 kilopascal
@@Thu Feb 8 18:58:12 2018	21.568627451 percent	959.5 revolutions_per_minute	14 kph	14.5098039216 percent	2.74509803922 percent	1585 second	22.7450980392 percent	87 degC	None	14 degC	101 kilopascal
@@Thu Feb 8 18:58:31 2018	23.5294117647 percent	658.5 revolutions_per_minute	8 kph	14.1176470588 percent	2.35294117647 percent	1604 second	22.7450980392 percent	86 degC	None	14 degC	101 kilopascal
@@Thu Feb 8 18:58:52 2018	22.3529411765 percent	1309.25 revolutions_per_minute	19 kph	17.2549019608 percent	5.49019607843 percent	1624 second	22.7450980392 percent	88 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 18:59:12 2018	36.862745098 percent	666.5 revolutions_per_minute	9 kph	14.9019607843 percent	2.74509803922 percent	1645 second	22.7450980392 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 18:59:32 2018	34.5098039216 percent	598.5 revolutions_per_minute	0 kph	14.1176470588 percent	2.35294117647 percent	1664 second	22.7450980392 percent	89 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 18:59:51 2018	16.0784313725 percent	1376.25 revolutions_per_minute	22 kph	15.2941176471 percent	3.52941176471 percent	1683 second	22.7450980392 percent	85 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:00:11 2018	43.137254902 percent	1584.25 revolutions_per_minute	19 kph	21.1764705882 percent	9.01960784314 percent	1704 second	22.7450980392 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:00:32 2018	14.1176470588 percent	1061.5 revolutions_per_minute	27 kph	13.7254901961 percent	2.35294117647 percent	1724 second	21.9607843137 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:00:51 2018	14.5098039216 percent	1396.5 revolutions_per_minute	34 kph	14.9019607843 percent	3.13725490196 percent	1743 second	21.9607843137 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:01:12 2018	41.568627451 percent	1920.25 revolutions_per_minute	28 kph	22.3529411765 percent	10.5882352941 percent	1764 second	22.7450980392 percent	88 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:01:32 2018	15.2941176471 percent	1160.5 revolutions_per_minute	39 kph	14.9019607843 percent	3.13725490196 percent	1784 second	22.7450980392 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:01:53 2018	39.2156862745 percent	1487.5 revolutions_per_minute	20 kph	20.0 percent	8.62745098039 percent	1805 second	22.7450980392 percent	88 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:02:12 2018	33.7254901961 percent	610.25 revolutions_per_minute	0 kph	14.5098039216 percent	2.74509803922 percent	1824 second	22.7450980392 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:02:32 2018	30.1960784314 percent	1396.75 revolutions_per_minute	34 kph	17.6470588235 percent	5.88235294118 percent	1844 second	22.7450980392 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:02:53 2018	15.2941176471 percent	993.0 revolutions_per_minute	33 kph	13.7254901961 percent	1.96078431373 percent	1865 second	22.7450980392 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:03:13 2018	33.3333333333 percent	598.75 revolutions_per_minute	0 kph	14.1176470588 percent	2.35294117647 percent	1885 second	22.7450980392 percent	89 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:03:32 2018	21.9607843137 percent	1137.75 revolutions_per_minute	28 kph	17.6470588235 percent	6.27450980392 percent	1905 second	21.9607843137 percent	85 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:03:53 2018	42.3529411765 percent	1130.25 revolutions_per_minute	34 kph	19.2156862745 percent	6.66666666667 percent	1925 second	21.9607843137 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:04:13 2018	32.5490196078 percent	1281.0 revolutions_per_minute	29 kph	17.6470588235 percent	5.88235294118 percent	1946 second	21.9607843137 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:04:34 2018	25.8823529412 percent	719.25 revolutions_per_minute	4 kph	14.1176470588 percent	2.35294117647 percent	1966 second	21.9607843137 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:04:54 2018	15.6862745098 percent	1102.75 revolutions_per_minute	27 kph	14.1176470588 percent	2.35294117647 percent	1986 second	21.9607843137 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:05:15 2018	13.7254901961 percent	1227.75 revolutions_per_minute	40 kph	13.7254901961 percent	1.96078431373 percent	2007 second	21.9607843137 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:05:35 2018	32.1568627451 percent	1212.0 revolutions_per_minute	39 kph	17.6470588235 percent	6.27450980392 percent	2027 second	21.9607843137 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:05:58 2018	7.05882352941 percent	1330.25 revolutions_per_minute	44 kph	14.1176470588 percent	2.74509803922 percent	2051 second	21.9607843137 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:06:19 2018	14.9019607843 percent	1254.75 revolutions_per_minute	42 kph	14.9019607843 percent	3.13725490196 percent	2071 second	21.9607843137 percent	86 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:06:39 2018	33.3333333333 percent	605.0 revolutions_per_minute	0 kph	14.5098039216 percent	2.74509803922 percent	2091 second	21.9607843137 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:07:02 2018	32.9411764706 percent	601.5 revolutions_per_minute	3 kph	14.5098039216 percent	2.74509803922 percent	2114 second	21.9607843137 percent	88 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:07:23 2018	11.7647058824 percent	1476.0 revolutions_per_minute	36 kph	16.0784313725 percent	4.70588235294 percent	2135 second	20.7843137255 percent	83 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:07:46 2018	28.6274509804 percent	1485.5 revolutions_per_minute	47 kph	16.862745098 percent	4.70588235294 percent	2158 second	20.7843137255 percent	87 degC	None	13 degC	101 kilopascal
@@Thu Feb 8 19:08:05 2018	32.1568627451 percent	612.5 revolutions_per_minute	0 kph	14.1176470588 percent	2.35294117647 percent	2177 second	21.9607843137 percent	86 degC	None	13 degC	101 kilopascal

Fig 1.2: A 10 minute snippet of the test data log produced from 50 minutes of driving. Sampled data logs all 10 PIDs in the spec, as well as 1 additional field (additional field was oil temperature, which was unsupported by the vehicle I was using to test).

4 GROUP CONCLUSION

Overall, we are on-track with project development, despite several setbacks that have significantly hampered the integration of all of our individually-developed systems into one program. Most of the individual components that comprise project Axolotl have been implemented, either in terms of complete functionality or in terms of an interface that will allow for much easier implementation of functionality at a later date. A significant amount of trial and error was experienced during development, not unexpected in such early stages of development. Approximately one-third to one-half of the requirements outlined in our software requirements specification have been fulfilled at this point in time. A large portion of our outstanding requirements can be fulfilled with relatively minor additions to our current codebase.

5 Group Activities by Week

5.1 Week 1

Completed navigation install; navigation system ready for testing. Work on media system started.

5.2 Week 2

Parts ordered. Work started on data logging system. Work started on UI started.

5.3 Week 3

Acquired parts from client. Navigation system and music player deployed on Jetson and demonstrated to TA. UI deployed on development computer and demonstrated to TA.

5.4 Week 4

Data logging, media system, and UI work continues. Acquired second set of parts from client.

5.5 Week 5

OBDII portion of data logging system completed. Data logging system tested and produces OBDII logs in the correct manner. Attempted initial integration, partial integration success. Data logging system is successfully executed from main GUI.

5.6 Week 6

Second attempt at integration; deployment on Jetson TX2 failed due as over-arching UI could not be installed due to incompatible Qt versions. Discovered that open-source Qt does not currently support Qt 5.10 on ARM platforms; UI written in Qt 5.10. Group elected to proceed into Alpha with individually-functioning components; plan to fully complete integration Week 7.