ENG 4000

Final Report
Automobiles Making Decisions (AMD 2)

Yahya Ismail

Chris Posca

Sean O’Brien

Manmeet Singh

Table of Contents

Table of Figures

Volume I - LIDAR Sensor Calibration Software Technical Volume

1.1 Executive Summary

The AMD2 LIDAR Calibration system was designed to combat the uncertainty created in autonomous vehicle LIDAR sensors in extreme weather conditions. Heavy rain, snow, and hail can make it very difficult for modern computer vision algorithms to operate as intended by creating noise in LIDAR results. This pressing issue in autonomous vehicles technology is just now becoming prevalent as major car manufacturers continue to test autonomous vehicles in northern climates.

While a solution to this problem is not out of the reach of current technologies, car manufacturers must strive for low cost solutions that can be implemented into a market ready vehicle. Therefore, the goal of this project was to use pre-existing resources on autonomous vehicles, effectively reducing the cost of the solution to development and maintenance.

The system operates by collecting information from sensors, specifically the optical cameras and precipitation sensors, and then deducing the correct values read by the LIDAR. The target for this analysis was license plates, being standardized and easily recognizable on any vehicle. By taking in an estimated precipitation density, the changing size of the plates over time, and the corresponding values from the LIDAR sensor the system can conclude the vehicles distance to surrounding targets and filter out any false readings from the LIDAR scan.

This system is lightweight, reliable, and only advances in these categories as the number of sensors increases. Catered towards reusability, classes are designed towards sensor types and therefore can use as many or as little sensors as possible assuming minimal requirements are met.

The future applications and scaling of this project are endless, as it can be adapted to discover other objects in a vehicles vision such as street lights or signs, providing every manufacturer a cost efficient 3rd party solution to their calibration needs.

1.2 Background Concepts

Autonomous navigation is an important field of research and, given the complexity of real-world environments, most of the systems rely on a complex perception system combining multiple sensors on board, which reinforces the concern of sensor calibration. Detection of some simple objects could potentially help in automatic sensor calibration within the complex system framework of Autonomous Vehicles. Data from multiple complementary sensors leads to increased accuracy and robustness which is why sensors need to be spatially and temporally calibrated. Calibration helps transform measurements from different sensors into a common reference frame making the vehicle safer and able to navigate more efficiently.

Currently, even though the Autonomous vehicle market is small(from a market capitalization perspective within the automobile industry landscape), most of the solutions focus on autonomous vehicle sensors to be calibrated to normal road conditions, traffic conditions, lane entry and exit scenarios, obstacle detection and avoidance, along with emergency braking circumstances. All these features are therefore calibrated as per normal weather conditions for daytime and nighttime, thereby leaving behind a massive opportunity to further understand the parameters required to calibrate sensors for harsh weather conditions such as heavy rain, fog, smog, snowstorm, hailstorm and sandstorms, to name a few.

During an event of harsh weather conditions, the calibrated sensors are unable to calibrate with the immediate surroundings thereby requiring the driver to operate the vehicle and henceforth eliminating the factor of autonomous driving. With the global autonomous vehicle market expected to reach $30 Billion by 2023, it is imperative to have a solution that can address different driving conditions and adapt to the local surrounding accordingly. The team identified this opportunity, and decided to implement a solution that can help major players within the AV Market landscape to calibrate their vehicles for harsh weather conditions, thereby opening up different markets for automobile OEMs.

After a lot of brainstorming and design problem iterations, the team was able to narrow down the scope of the initially defined problem. The problem statement of the capstone project is as follows:

" *To calibrate sensors of an autonomous vehicle to ensure reliability and full interoperability in harsh weather conditions such as snow-storms, Sandstorms, etc.*"

This problem statement then helped the team to list out competencies required to achieve the scope of the proposed problem. The following sensor areas were determined to be crucial to system/solution success:

1. LIDAR Sensor Class
2. Camera Sensor Class
3. Precipitation Density Sensor Class

LIDAR Sensors

LIDAR Sensors (Light Detection and Ranging) provides an incredibly-detailed 3D Representation of the car's surroundings. Pulses of light are emitted from the lasers which spin in circle. When they hit an object such as a tree, they bounce back to the Lidar sensor. Distance is measured based on how long it takes the pulses of infrared light to travel back. The results are then compiled into a point cloud, which can be defined as a set of data points in space. This happens millions of times per minute, which means millions of data points and calculations are taking place at incredible speed. The result is a constantly-updated map of the world around the car as shown in Figure 1.

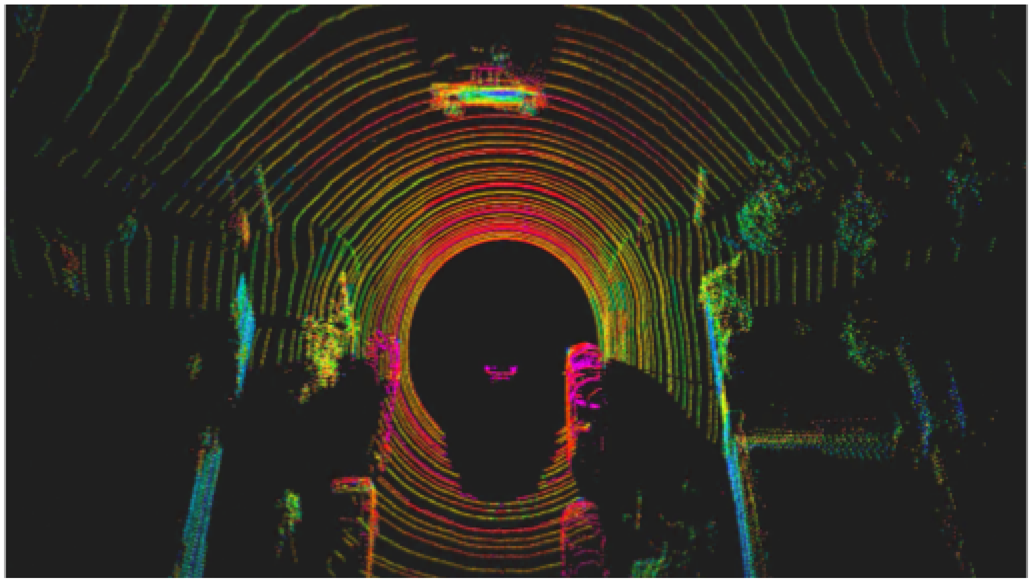

Figure 1 - Point cloud of the immediate surrounding generated using on-board LIDAR sensor (Voyage)

These sensors however become highly ineffective during the times of harsh environment such as Snowstorms, heavy rainfall etc, when the pulse of infrared light bounces back to the LIDAR sensor after making contact with the snow/rain particle.

Camera Sensors

Camera sensors are installed on the autonomous vehicles to help collect data regarding different road conditions, road lanes, traffic signs among other things. With the camera resolution become better every day with emerging technological breakthroughs, cameras provide more reliability in collection of data required to make crucial driving decisions. An example of such system can be seen in Figure 2.

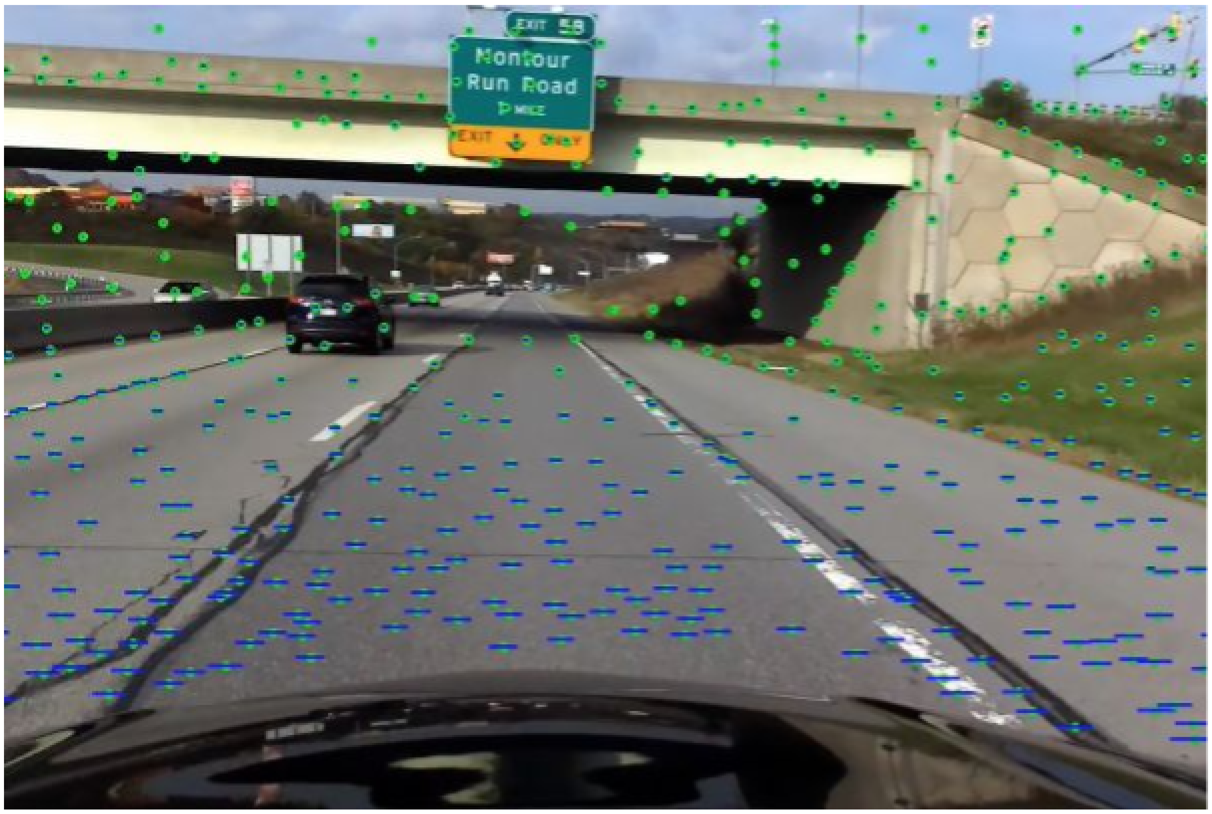

Figure 2 - Images Captured using on-board vehicle camera (Seegrid)

Precipitation Density Sensors

This class of sensors helps the vehicle to determine the relative humidity in the immediate surrounding of the vehicle, thereby providing an indication of different weather conditions such as rain, snow etc.

Figure 3 - High Relative humidity in the immediate vehicle surrounding

The proposed solution proposes to utilize the precipitation sensor and the camera to determine when the LIDAR is providing incorrect values to the system. This would be done using the front-facing camera to estimate the distance from the number plate of the closest vehicle. The distance can be estimated by recognizing the license plate of the car and calculating the pixel size of the plate, inferring the plate's distance from the camera as the system has the capability to approximate dimensions of a license plate by using a machine learning algorithm to learn. Once the computer onboard the vehicle has an estimate of how far the closest vehicle is, it can reject outlier data caused by floating objects like snowflakes getting in the way of the light rays.

1.3 User Requirements

For the successful application of this product, users should already have a system which provides the following. However, a brief outline of essentials is below:

1.3.1 Software Requirements

- OpenCV 2.4.13.6-1 - for basic computer vision operations
- Tesseract 4.0.0-1 - for character identification
- Openalpr 2.3.0-1 - for license plate identification

Note: Software Requirements are selected based on the dependencies of related programs and may not reflect ideal versions for each respective package. Future maintenance will be required to update the selected libraries, however this list reflects the versions used in development.

1.3.2 Hardware Requirements

- Front facing camera
- Front bumper camera*
- Rear facing camera*
- Windshield precipitation sensor*

* Sensors are not necessary for operation but greatly increase reliability and reduce computation time.

The cameras required do not necessarily need to meet a certain specification, however it is assumed that these cameras exist in a pre-existing autonomous vehicle. As a minimum recommendation 30 fps and 1920x1080 resolution is sufficient, however the system will benefit from more frames and larger resolutions as increased fps cuts the system response time and higher resolution increases system accuracy.

1.4 System Requirements

The requirements as revised and verified can be seen in Figure 4 below.

Figure 4

Requirement	**Verification Results**
The software shall identify license plates in the line of sight of camera sensors.	The software identifies license plates with an acceptable certainty, and differentiates license plates from similar objects.
The software shall identify license plates regardless of irregular size, colour, or viewing angle.	The software attains comparable certainty when identifying irregular license plates.
The software shall infer distances between any identified license plates and the camera which identified the plates.	This aspect of the project has not yet been tested.
The software will utilise distance approximations to correct interference in LIDAR sensors.	This aspect of the project has not yet been tested.
The software shall not use more than 10% of the Vehicle's CPU time.	Design is as lightweight as possible, and it will be clear whether or not the requirement id verified when the product is completed.
The software	This is enforced by the design of our software, so it would be fair to

shall produce a decision every 500ms.	say this is verified.
The software shall run on the hardware present in the vehicle.	The components of the project run on the testing rig independently.
The software shall receive input from the various sensors.	Since the reliability of the sensors are external to the system, only the presence of a reliable signal is a concern. This is a precondition to the operation of the software.

1.5 System Architecture

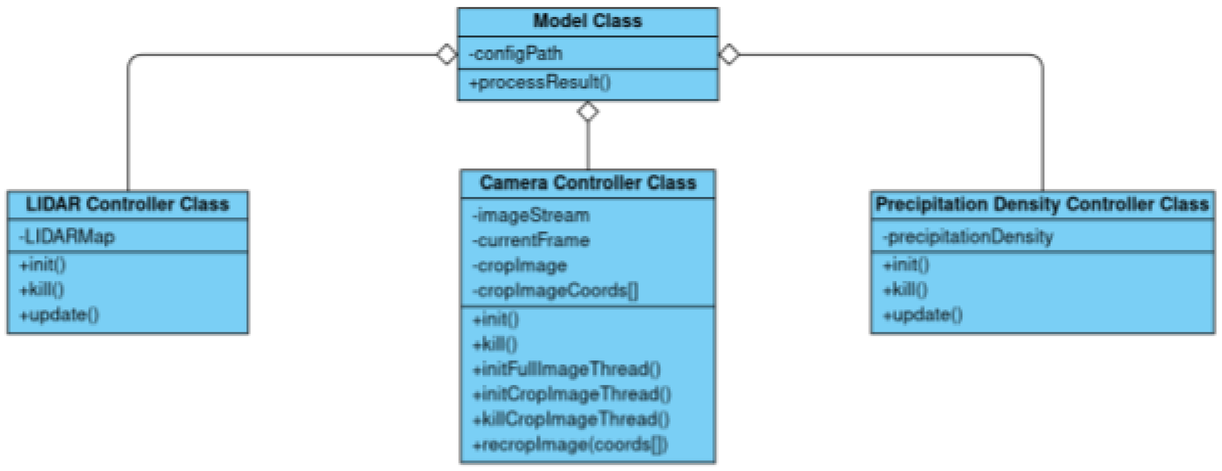

Figure 5 - UML Diagram for Sensor Calibration System

Above in figure 5 is the UML diagram of the architecture. The main class is the model class, which is responsible for instantiating the other classes and maintaining the system. It also produces the response from the logic of the system. The three controllers that can be instantiated are the LIDAR controller, camera controller, and the precipitation controller. Their full list of private and public attributes and operations can be seen above, to give the reader an intuition into the system's structure.

1.5.1 LIDAR Controller Class

This class is responsible for receiving and handling the LIDAR data. Designed with the assumption of pre-processed data fed from an operating system, this class will act as an accessible object holding all information retrievable necessary to the system. This information can be accessed by the model class when necessary.

1.5.2 Camera Controller Class

The camera class is responsible for processing video streams received from the operating system. This class has multiple threads which balance the detection of license plates as well as the processing of selected frames such that the program can track it's target. There are two main detection threads within this class.

The first thread is responsible for detecting license plates via openalpr on a full scale image, fed directly from the camera. Periodically this thread will activate itself and perform its operation, updating the coordinates of the target on the image plane. The thread should be considered a "fail proof" detection, and the absence of a target or inability to properly detect a target will re-initialize this thread. It is also responsible for the initialization of any other threads created by this class. Therefore, in the event of another thread's failure, the class will loop to the beginning of this full scale search, once again detecting a point at which the program can operate.

Once the primary thread has successfully executed, a secondary thread is launched in parallel with it. The responsibility of this thread is to detect a pre-processed image at as low an execution time as possible. This thread will crop the image, degrade the resolution, and transform the image before detection. It will also create some sort of bounds for motion error within a calculated pixel range and re-adjust the crop with every new iteration. In the event of a failure to detect a previously known target, this thread will terminate itself until the execution of a new primary thread is complete, restarting it's life cycle. This will give the controller the opportunity to find lost targets, adjust in the event of an error and most of all provide a reliable result.

The class is isolated from the performance and specifications of the camera, making it operable on any image stream. The only drawback of low grade hardware would be increased delay between sampling period and execution. This design greatly excels in reusability, as it can be adapted to as many cameras as desired.

1.5.3 Precipitation Density Controller Class

The precipitation density controller class is responsible for obtaining data from vehicle's windshield monitor. Modern vehicles are equipped with some sort of infrared sensor mounted on the rear-view mirror or on the top of the dashboard. These sensors are used to estimate the speed at which the windshield wipers should operate by estimating precipitation density. By retrieving this information, the class can pass a noise estimation parameter to the model, which can contribute to the accuracy of the algorithm.

1.5.4 Model Class

The model class is the main class in the system. Via a configurations file, the model class begins its operation by instantiating all available sensors, and initializing the controller objects. Once all of the controllers are available and running, the model will acquire any necessary information and correct any noisy points. The performance of this correction will vary with incoming data from the multiple sensors, in a weighted linear regression fashion.

1.6 Non-conformances From Test Procedures and Their Current Disposition

At the time of this document's creation, the test procedures are verified unless stated otherwise. Those which have not been verified are delayed for developmental reasons. Those which are verified have already been done so in the test review.

1.7 Self-evaluation Regarding Technical Pack Performance

A large consideration in the design of this application was the running time, memory management, and it's performance on hardware. Unfortunately since the algorithm is designed to be multiplatform, the actual performance of the system will partially depend on the performance of the machine it is run on. Nonetheless, the system was tested on developer's systems to demonstrate the possibility of the real world usefulness of the system, as well as any efforts made in the design aspect to shave off operating costs.

1.7.1 Running Time

The running time of the algorithm is the biggest concern with the program. It is essential that a calibration event must execute as quick as possible, giving notice to external systems and reducing the probability of system failure. By resolving each individual controller's state in a reasonable time period, the model is given enough time to appropriately calculate any changes that must be made to LIDAR map data. Since the reading of the LIDAR and precipitation density is trivial (solely depending on the response time of sensors, and therefore external to the system), the large focus for this aspect was the camera controller.

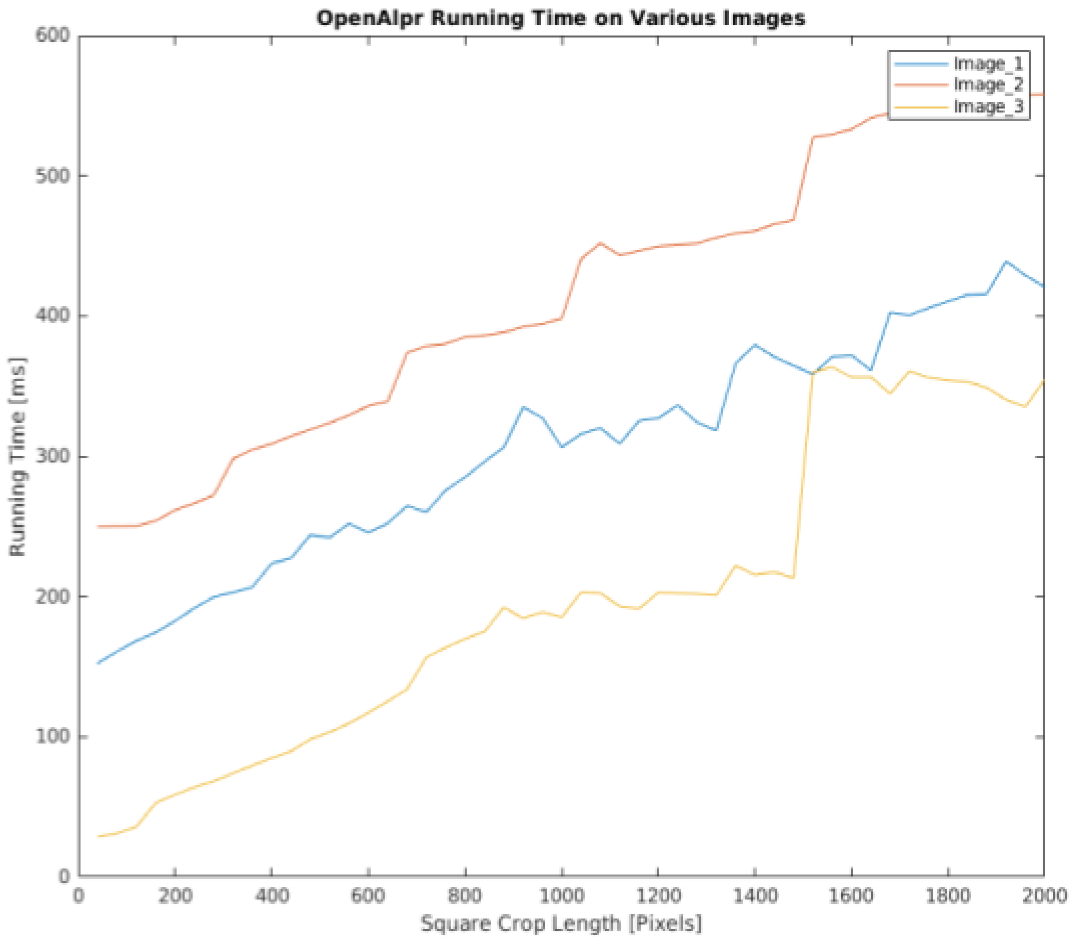

Figure 6 - Openalpr Running Time Test

As can be seen by figure 7, the running time of the openalpr algorithm dramatically decreases as the size of the frame decreases. At smaller resolutions, the need for complex operations on the images decreases.

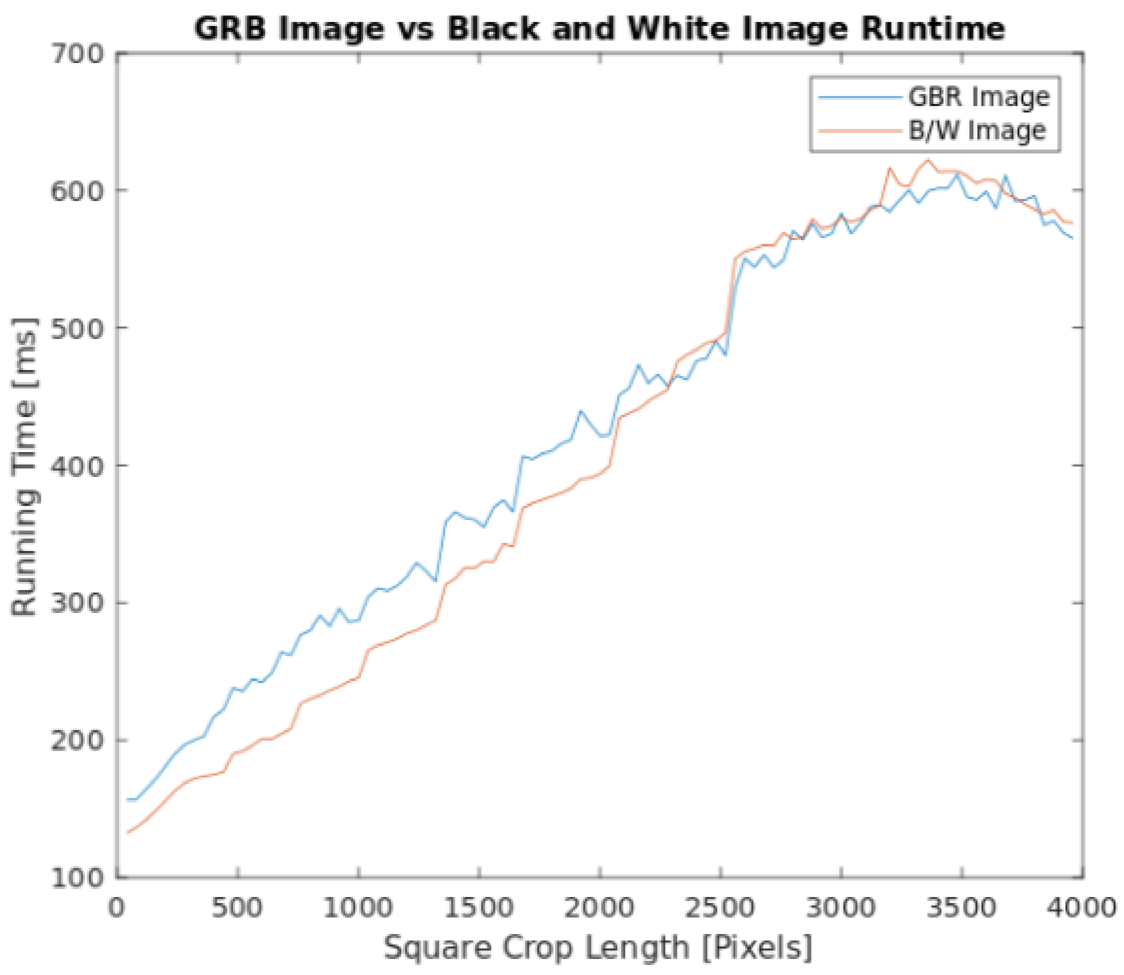

Figure 7 - Pixel Color Format Comparison on Openalpr Runtime

In figure 7, the effect another small performance tweak can be seen, switching from GRB to black and white. As can be seen by the graph, in smaller samples, where targets may be more easily detected or ignored, the algorithm performs better. As the image size increases, including the complexity of the images the algorithm is analyzing, the cost goes up. This created a situation where the algorithm performed better in high light settings and therefore the color could be sacrificed, however in a larger, lower light, and possibly more diverse image, the algorithm had struggles with black and white. To shave off execution time, the algorithm will detect when it is suitable to use a black and white image, and when it should convert if ever.

1.7.2 Memory Management

The algorithm requires little to no memory use above that which openalpr requires. In its current form openalpr calls for a minimum requirement of 4GB of memory. And while this is excessive, this is under the assumption of a full fledged operating system requirement. Therefore while it is safe to assume 4GB would be sufficient to run this system, it is quite possible that less may work as well.

1.7.3 Hardware Performance

The hardware performance of this product is largely untested due to the budgetary constraints. However, the libraries used in this system are multiplatform and should have no issues running on any hardware available today. Additionally, all handling of threads is directly handled by the operating system kernel and therefore is fully optimized on all platforms. In the event that this changes, the system shall be altered. However, at this current date foundational C libraries such as pthreads have not been altered and will most likely continue to be supported as the most computationally frugal building blocks of programming languages.

1.8 Strengths and Weaknesses of Solution

The design of this system was created from the premise that no extra hardware would be essential to solve the problem. This restriction allowed for the design to be cost free, solely operating off of hardware that would already exist in autonomous vehicles. The algorithm makes use of a standardized marker that can be found on the front and rear and all vehicles, and can be easily adapted from country to country with a small amount of training data. Also, the programming uses fundamental C libraries that can be run on any operating system available today. When considering these aspects, the design is exactly what was desired. The technology is reusable, platform and hardware independent, cost-free and easily accessible. While the system fulfils the constraints of success above, this also contains a fair amount of drawbacks.

A large concern in this design is that the computation time can only truly be determined under the load of its respective operating system. Since multithreading is dispatched by the kernel of the operating system, the algorithm may be bottlenecked by the execution of the platform it is running on. The platform may also be largely dependent on the capabilities of the hardware provided. This creates an uncertainty in the true performance of this system.

The software in this current iteration suffers from the fact that it only identifies license plates. In the event that no other vehicles surround the user, there is no reference for this program to work. Thus, the solution provided can only be seen as a partial solution at best. A more costly and possibly hardware intensive solution is definitely a possibility, but this would defeat the purpose of this product.

Finally, since a large amount of libraries are from open source projects, it is impossible to verify that the program will safely execute without testing the underlying systems. This ultimately means that the program can never be used in a critical system without extensively testing these libraries.

1.9 Engineering Budgets

Fortunately, for the development of this program a budget was not necessary. All of the software libraries used in the design of this program are open-source. With an estimated development cost of four months of salary for two junior engineers at $60000 CAD, the development cost of this program would cost about $40000 CAD. Considering the small scale of the program, and the estimated maintenance cost of around 20%, the program should only cost around $8000 CAD annually to maintain.

Volume II - Testing Environment Technical Volume

2.1 Executive Summary

The AMD2 Testing Environment was designed to provide a reasonably accurate and cost efficient process to test autonomous vehicle sensor calibration methods. Under the requirements that the team was initially proposed with the budget granted, real world testing was impossible. From this, the idea to build a testing environment came to fruition.

The system was designed specifically for the targeted software. With this in mind variability on driving conditions, weather conditions and environment conditions were an essential part of the design. The system allows access to multiple cameras placed on vehicles where they might realistically be placed on autonomous vehicles on the market today: the front and rear bumper, mounted to the rear view mirror, etc. There is also access to other sensor types such as LIDAR by performing ray tracing from these camera points.

The system provided our development team the ability to test the algorithms we created without the necessity of hardware. Calibration software can be run on the platform natively, and so simulated variables can be directly accessed. This also allows for the potential to compare results from our calculations with the real values such as distance or speed that can be measured in our simulation, and apply changes to benefit the outcome.

This design allows our system to fill a specific niche in the market, targeted at companies who develop software but may not have trivial access to expensive equipment. It can also be seen as a cost free platform for testing, providing a risk free solution to a large company who sees no reason to waste capital on preliminary tests. While this platform should never be seen as a reliable representation of reality, it can provide a great deal of insight on how systems may behave in the real world.

2.2 User Requirements

Our simulation environment is built to be stable under many different systems and the minimum recommended requirements are fairly conservative .

2.2.1 Software Requirements

In order to run the simulation, it is required that the user satisfy the following software requirements.

Operating System: Windows 7 SP1+, maxOS 10.11+, Ubuntu 12.04+, SteamOS+

2.2.2 Hardware Requirements

In order to run the simulation, the following are the minimum hardware requirements.

CPU: SSE2 instruction set support

Graphics card* with DX10 (shader model 4.0) capabilities

* A dedicated graphics card is not recommended but not required. Many modern CPUs come with embedded graphics chips that can satisfy this requirement.

2.3 System Requirements

Figure 8 - Testing Environment System Requirements

Requirement	**Verification Results**
The simulation environment must follow a model which mirrors the laws of physics	The environment was created in unity, which has a built in physics model. Unity is designed to be realistic and mirror reality, giving a solid foundation for this requirement. Of course, a perfect outcome is impossible, but this is the best available option with current technology.
The simulation environment must be able to simulate vehicles	Vehicles have been successfully created and simulated in the environment.
Optical sensors must be mounted to vehicles	The environment is capable of creating new cameras which can be programmed to certain positions. The sensors can successfully be mounted to cars.

LIDAR sensors must be mounted to vehicles	The environment is not specifically capable of creating LIDAR sensors. This can be simulated by using ray tracing. This operates under the assumption that the speed of light is negligible. This is not absurd relative to the speed and quality expected and therefore this is acceptable.
The user must have access to manipulate control variables within the system	By design of the API, the user can manipulate control variables and therefore this is verified.
The user must have access to read dependent variables created within the system	By design of unity, measurements can be made during any instance of time in the simulation, and therefore this is verified.
The environment must be capable of simulating extreme weather conditions	By attaching pre-existing weather options to the designed API, this requirement is satisfied.
The system must contain an intuitive api to control the operation of the system.	The API has been built, and so this has been satisfied.

2.4 System Architecture

The idea behind the testing environment is that it is a completely separate entity from the actual sensor calibration solution. The simulation runs in its own environment and is called via its public facing API, but outside of this it has no lines of communication to the sensor calibrator. The reason that the architecture of the simulation is so loosely coupled is because it presents the business opportunity of developing two equally profitable products targeting different markets.

As shown in our architectural diagram in figure 9, the simulation runs in its own environment from the controller, and exposes an API layer that the controller may use. This API layer is used to control the users car in the simulation. The simulation is then displayed on the primary monitor, where a camera will take the monitor as an input for the controller. In this sense, the architecture acts as a feedback loop, working to stabilize the users car over time. Data and information such as LIDAR, RADAR, and precipitation meters will be outputted to a seperate file that may be read from the controller asynchronously to the main loop.

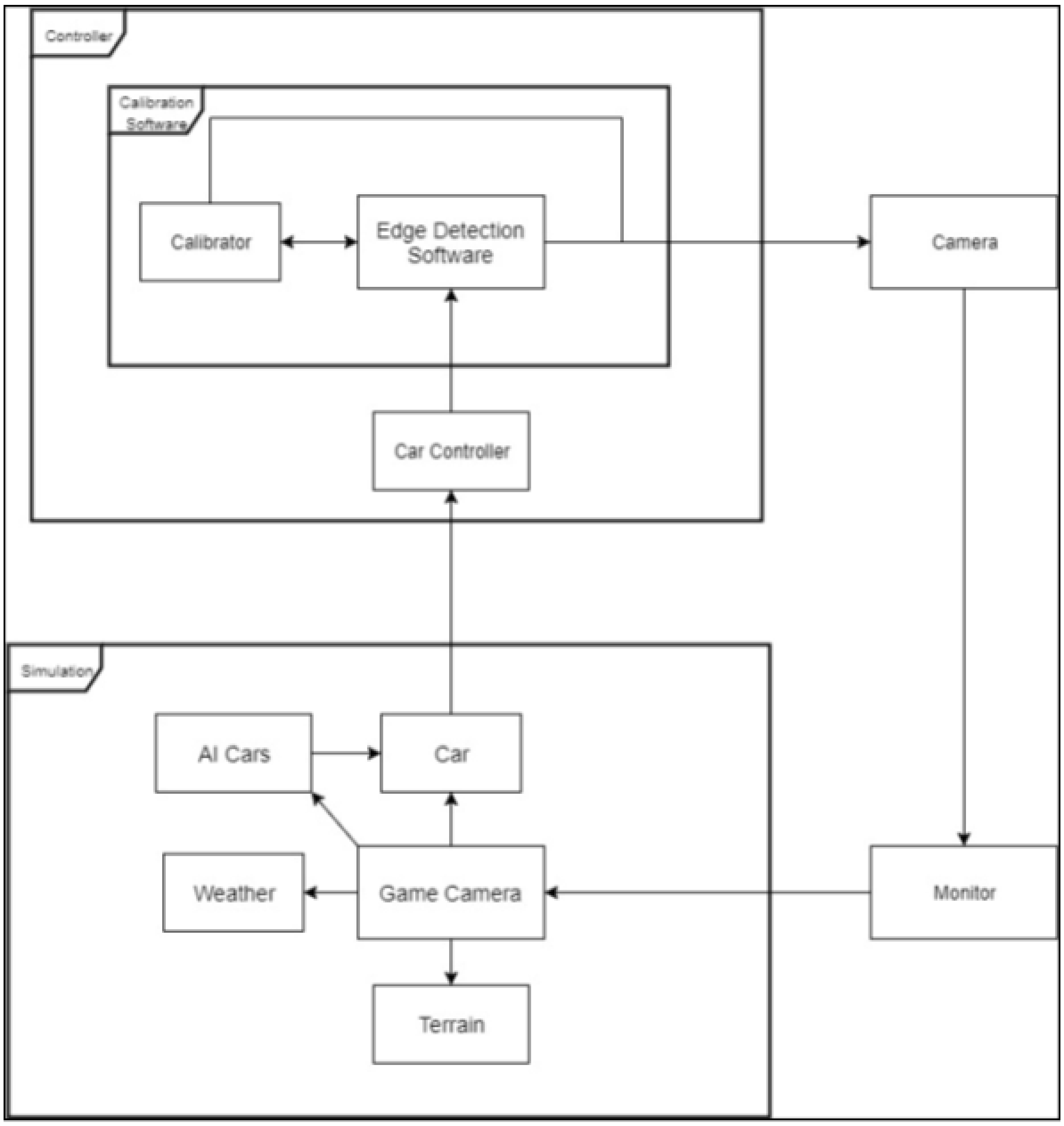

Figure 9 - Testing Environment Architecture

2.5 Strengths and Weaknesses of Solution

The simulation provides the unique testing advantage of zero cost and zero capital testing while still providing reasonable results. By making use of accurate models of real world physics the simulation can be a great alternative to real world tests. Also, the fact that this is a digital simulation gives to opportunity for exhaust testing, in which randomized events can occur over a near infinite amount of repetitions. The complex yet easily managed API allows for quick setup and a large outcome space.

The only issue with a simulation with simulation testing is that no engineer should ever replace real world testing with simulations. This form of testing is designed to test prototypes and so it is limited in capabilities. It can not and most likely will never fully replicate real world testing.

In the future the only improvements that the simulation can hope to achieve are improving the accuracy of its physics. Unfortunately, this is usually a performance issue and not a software issue, held back by the capabilities of modern hardware. As hardware improves in the future, models will have to be designed to scale well and mirror reality as best as possible.

2.6 Engineering Budgets

The only dependency which was not designed by the team was Unity, a popular free* game engine. All other software was designed by the development team. With an estimated development cost of four months of salary for two junior engineers at 60000 CAD, the development cost of this program would cost about 40000 CAD. Considering the small scale of the program, and the estimated maintenance cost of around 20%, the program should only cost around 8000 CAD annually to maintain. Once a $100,000 USD profit margin is exceeded, the annual operating cost for unity is increased to $1500/seat USD annually. This brings operating totals to approximately $12000 CAD.

Volume III - Management

3.1 As-built Work Breakdown Structure

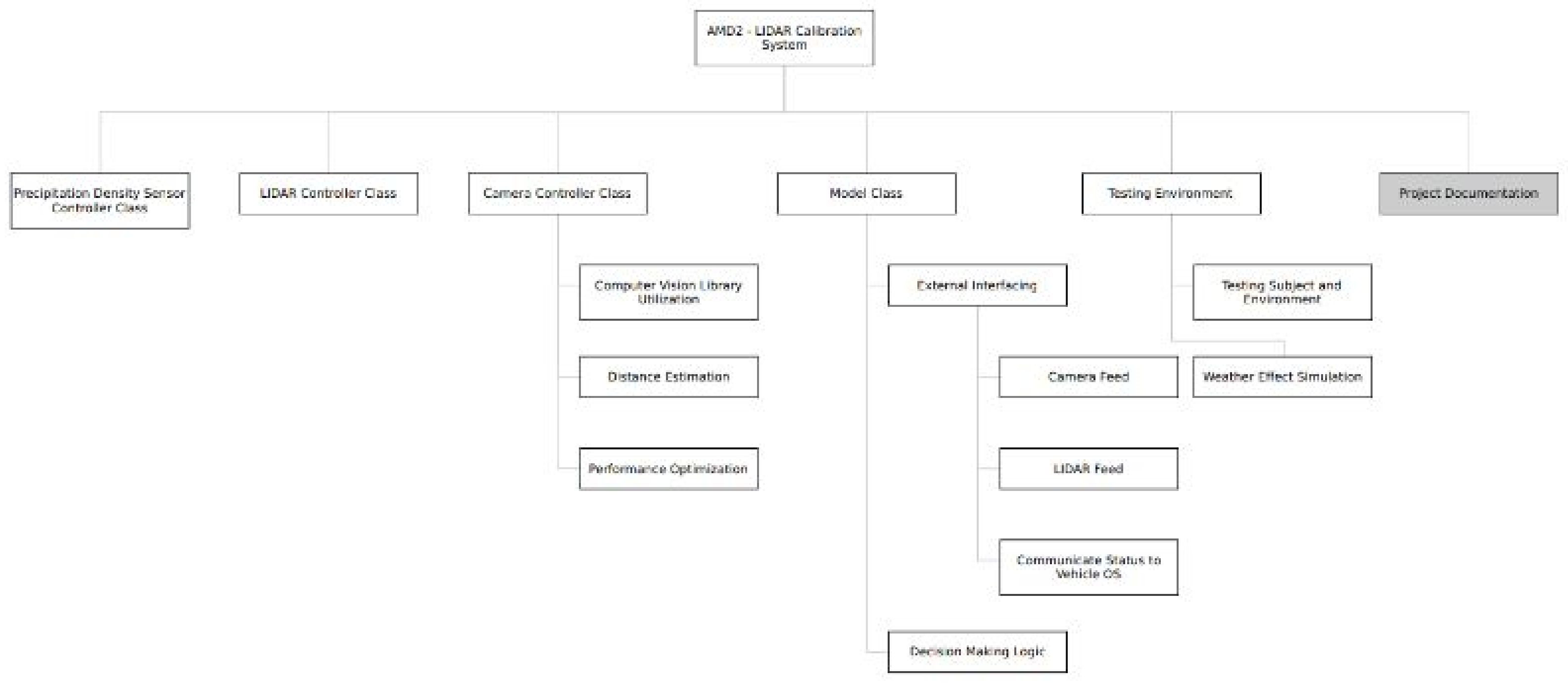

Figure 10 - Work Breakdown Structure

3.2 As-built Work Package Description

WP ID: 1.1	**WP Name:** Computer Vision Library Utilization
Expected Effort in Hours: 15-20	**WP Manager:** Chris - 10 hours **Support:** Sean - 10 hours
Expected Start Date: Jan 29	**Expected End Date:** Mar 5
Inputs Needed: - Personal Computer - Internet Connection - OpenAlpr manuals and tutorials for utilized languages (C/C++) (all documents, materials etc., required to perform the work package)	
Tasks to be performed: - Acquire background knowledge necessary encorporate OpenAlpr libraries into a C++ project - Integrate Library onto testing PC - Perform initial tests on libraries to verify that they suit our purposes	
Outputs Generated: - Working access to OpenAlpr libraries (all systems/subsystems/documents or other outputs from this work package)	

WP ID: 1.2	**WP Name:** Distance Estimation
Expected Effort in Hours: 15-20	**WP Manager:** Sean - 15 hours **Support:** Chris - 5 hours
Expected Start Date: Jan 29	**Expected End Date:** Mar 5
Inputs Needed: - Computer vision and optics research - C++ experienced developer	
Tasks to be performed: - Write C++ function to parse JSON file produced by OpenAlpr library and store pixel locations of license plate. - Write C++ program to perform distance estimation on pixel coordinates of plate in image using estimate of object size - Write C++ program to utilize camera parameters as well as pixel size and estimate of actual object dimensions	
Outputs Generated: Class to be instantiated in the Camera Controller Class, for use in the Model Class	

WP ID: 1.3	**WP Name:** Performance Optimization
Expected Effort in Hours: 15-20	**WP Manager:** Chris - 20 hours **Support:**
Expected Start Date: Mar 5	**Expected End Date:** Apr 26
Inputs Needed: - Working Camera Controller Class - OpenAlpr library implementation - Working Distance Estimation Class	
Tasks to be performed: - Improve model class performance time (repeats every 500ms) and reduce computational requirements (should run on laptop) - Minimize memory accesses and use optimal C++ functions - Minimize code length by reusing segments	
Outputs Generated: - Model Class which performs up to the standards set out in the requirements	

WP ID: 2.0	**WP Name:** LIDAR Controller Class
Expected Effort in Hours: 10	**WP Manager:** Chris - 7.5 hours **Support:** Sean 2.5 - hours
Expected Start Date: Mar 5	**Expected End Date:** Apr 26
Inputs Needed: - LIDAR sample data	
Tasks to be performed: - process LIDAR data to produce an estimate of the distance between the vehicle and the object closest to the front facing camera	
Outputs Generated: - Program to process LIDAR data and produce distance estimate, which will then be checked against the distance estimate produced by the calibration software - An interface that the model can use to access the LIDAR data stream in a suitable format	

WP ID: 3.0	**WP Name:** Precipitation Density Sensor Controller Class
Expected Effort in Hours: 2	**WP Manager:** Chris - 2 hour

	Support:
Expected Start Date: Mar 5	**Expected End Date:** Mar 10
Inputs Needed: - Precipitation density measurement	
Tasks to be performed: - retrieve precipitation reading from sensor and return it to the Model class	
Outputs Generated: - An interface that the model can use to access precipitation density in a suitable format	

WP ID: 4.1.1	**WP Name:** Camera Feed
Expected Effort in Hours: 10-15	**WP Manager:** Sean - 10 hours **Support:**
Expected Start Date: Jan 29	**Expected End Date:** Mar 5
Inputs Needed: - Live feed from webcam - Memory to store results	
Tasks to be performed: - In a C++ class, sample the feed from the camera when prompted by the camera controller class	
Outputs Generated: - Ability for to be provided to the Camera Controller Class by the Model Class	

WP ID: 4.1.2	**WP Name:** LIDAR Feed
Expected Effort in Hours: 10-15	**WP Manager:** Chris - 5 hours **Support:** Sean - 2 hours
Expected Start Date: Jan 29	**Expected End Date:** Mar 5
Inputs Needed: - LIDAR map from sensor	
Tasks to be performed: - In a C++ class, build an interface for the LIDAR data received	
Outputs Generated: - A class which will handle incoming sensor data from LIDAR sensors	

WP ID: 4.1.3	**WP Name:** Communicate Status to Vehicle Operating System
Expected Effort in Hours: 8-10	**WP Manager:** Chris - 5 hours **Support:**
Expected Start Date: Feb 20	**Expected End Date:** Mar 5
Inputs Needed: - Completed software model logic	
Tasks to be performed: - Create a standardized output from the model which will effectively communicate the status of the software	
Outputs Generated: - A useful return from the model software - General user operation guidelines for the software	

WP ID: 4.2	**WP Name:** Decision Making Logic
Expected Effort in Hours: 15-20	**WP Manager:** Chris - 50 hours **Support:** Sean - 15 hours Yahya - 10 hours
Expected Start Date: Feb 14	**Expected End Date:** Apr 25
Inputs Needed: - All controller classes completed - Research about execution strategies completed	
Tasks to be performed: - Write the c++ main logic class	
Outputs Generated: - Final calibration system project completion	

WP ID: 5.1	**WP Name:** Testing Subject and Environment
Expected Effort in Hours: 15-20	**WP Manager:** Yahya - 20 hours **Support:**
Expected Start Date: Jan 14	**Expected End Date:** Feb 28
Inputs Needed:	

Tasks to be performed: - create a realistic automobile simulation environment - create an easily controllable API to edit future simulation constraints
Outputs Generated: - a platform for automobile simulation which can natively run calibration software and is configurable for future tests

WP ID: 5.2	**WP Name:** Weather Effect Simulation
Expected Effort in Hours: 8-10	**WP Manager:** Yahya - 5 hours **Support:**
Expected Start Date: Jan 29	**Expected End Date:** Feb 28
Inputs Needed: - Testing environment	
Tasks to be performed: - develop a simulation to test the calibration software	
Outputs Generated: - randomly generated simulation with control variables such that adequate tests can be conducted	

WP ID: 6.1	**WP Name:** Team and Project Summary
Expected Effort in Hours: 10	**WP Manager:** Manmeet - 5 hours **Support:** Sean - 1 hour Chris - 1 hour Yahya - 1 hour
Expected Start Date: Sept 13	**Expected End Date:** Sept 18
Inputs Needed: - Statement of Work - Team member CVs - Background research on the topic and market	
Tasks to be performed: - Define the project objectives - Identify stakeholders - Introduce team members and capacity - Explore product market and management strategy	
Outputs Generated: - Team and Project Summary for peer review	

WP ID: 6.2	**WP Name:** Requirements Review
Expected Effort in Hours: 20	**WP Manager:** Manmeet - 5 hours **Support:** Sean - 5 hours Chris - 5 hours Yahya - 5 hours
Expected Start Date: Sept 18	**Expected End Date:** Sept 26
Inputs Needed: - Meeting with customer/sponsor - Research into statement of work - Notes from brainstorming meeting with group members	
Tasks to be performed: - Accurately capture the essence of the problem using a list of requirements for a solution to the problem - Create functional, performance, interface, regulatory and programmatic requirements to capture the entire problem	
Outputs Generated: - A requirements review for grading and peer review	

WP ID: 6.3	**WP Name:** Agile Roadmap
Expected Effort in Hours: 15	**WP Manager:** Manmeet - 7.5 hours **Support:** Chris - 5 hours Yahya - 2.5 hours Sean - 2.5 hours
Expected Start Date: Oct 2	**Expected End Date:** Oct 16
Inputs Needed: - Requirements review feedback - Agile development experienced members - Product vision	
Tasks to be performed: - Define the product vision and the timeline of features to include in the product - Create organization tools like spreadsheets and tables for tracking Sprint results and progress towards the final product - Write out the first sprint plan	
Outputs Generated: - Agile roadmap for grading and peer review	

WP ID: 6.4.1	**WP Name:** Sprint 1 Report
Expected Effort in Hours: 15	**WP Manager:** Yahya - 7.5 hours **Support:** Chris - 7.5 hours
Expected Start Date: Oct 17	**Expected End Date:** Oct 30
Inputs Needed: - Agile roadmap feedback - Scrum records - Meeting minutes	
Tasks to be performed: - Document weekly scrum records - Write Sprint Retrospective - Plan next sprint	
Outputs Generated: - Sprint Review Report for grading and peer review	

WP ID: 6.4.2	**WP Name:** Sprint 2 Report
Expected Effort in Hours: 15	**WP Manager:** Yahya - 5 hours **Support:** Chris - 5 hours
Expected Start Date: Oct 31	**Expected End Date:** Nov 13
Inputs Needed: - Previous sprint feedback - Scrum records - Meeting minutes	
Tasks to be performed: - Document weekly scrum records - Write Sprint Retrospective - Plan next sprint	
Outputs Generated: - Sprint Review Report for grading and peer review	

WP ID: 6.4.3	**WP Name:** Sprint 3 Report
Expected Effort in Hours: 10	**WP Manager:** Sean - 10 hours
Expected Start Date: Nov 14	**Expected End Date:** Nov 27
Inputs Needed: - Previous sprint feedback	

- Scrum Records - Meeting minutes
Tasks to be performed: - Document weekly scrum records - Write Sprint Retrospective - Plan next sprint
Outputs Generated: - Sprint Review Report for grading and peer review

WP ID: 6.4.4	**WP Name:** Sprint 4 Report
Expected Effort in Hours: 15 hours	**WP Manager:** Sean - 10 hours **Support:** Manmeet - 5 hours
Expected Start Date: Jan 7	**Expected End Date:** Jan 29
Inputs Needed: - Previous sprint feedback - Scrum Records - Meeting minutes	
Tasks to be performed: - Document weekly scrum records - Write Sprint Retrospective - Plan next sprint	
Outputs Generated: - Sprint Review Report for grading and peer review	

WP ID: 6.4.5	**WP Name:** Sprint 5 Report
Expected Effort in Hours: 15	**WP Manager:** Sean - 7.5 hours **Support:** Yahya - 2.5 hours Chris - 2.5 hours
Expected Start Date: Jan 31	**Expected End Date:** Feb 21
Inputs Needed: - Previous sprint feedback - Scrum Records - Meeting minutes	
Tasks to be performed: - Document weekly scrum records - Write Sprint Retrospective - Plan development leading into test review	
Outputs Generated: - Sprint Review Report for grading and peer review	

WP ID: 6.5	**WP Name:** Test Review Report
Expected Effort in Hours: 40	**WP Manager:** Sean - 20 hours **Support:** Chris - 20 hours
Expected Start Date: Feb 22	**Expected End Date:** Mar 5
Inputs Needed: - Viable, testable product	
Tasks to be performed: - Operate tests on the as-built product - Record test results and create visuals to display said results - Make conclusions about the fulfillment of system requirements	
Outputs Generated: - Document clearly demonstrating the performance of the testing via intuitive figures - Generate a report indicating the understanding of the fulfilled requirements, and steps going forward to fulfill remaining requirements	

WP ID: 6.6	**WP Name:** Final Report
Expected Effort in Hours: 35-40	**WP Manager:** Chris - 20 hours **Support:** Sean - 20 hours Manmeet - 5 hours Yahya - 5 hours
Expected Start Date: Mar 26	**Expected End Date:** Apr 2
Inputs Needed: - Late development stages of projects - List of references and relevant work information recorded during development	
Tasks to be performed: - Document the timeline and management of the project - Compile figures which quickly and effectively convey information	
Outputs Generated: - Final documentation	

3.3 As-built Resource Allocation Matrix

To summarize, each member's contribution throughout the process, the tables below represent the number of hours of work performed by each individual along with the work package IDs associated with it:

September 2018		
Name	WP ID	# of hours
Chris	Total 6.1/6.2	6 1/5
Sean	Total 6.1/6.2/6.4.4	16 1/5/10
Manmeet	Total 6.1/6.2/6.4.4	15 5/5/5
Yahya	Total 6.1/6.2	6 1/5

October 2018		
Name	WP ID	# of hours
Chris	Total 6.3/6.4.1	12.5 5/7.5
Sean	Total 6.3	2.5 2.5
Manmeet	Total 6.3	7.5 7.5

Yahya	Total 6.3/6.4.1	10 2.5/7.5

November 2018		
Name	WP ID	# of hours
Chris	Total 6.4.2	5 5
Sean	Total 6.4.3	10 10
Manmeet	-	-
Yahya	Total 6.4.2	5 5

December 2018		
Name	WP ID	# of hours
Chris	-	-
Sean	-	-
Manmeet	-	-
Yahya	-	-

January 2019		
Name	WP ID	# of hours
Chris	-	-
Sean	Total 6.4.4	10 10
Manmeet	Total 6.4.4	5 5
Yahya	Total 5.1	10 10

February 2019		
Name	WP ID	# of hours
Chris	Total 1.1/1.2/1.3/4.1.2/4.2/6.4.5/6.5	67.5 10/15/20/5/10/2.5/5
Sean	Total 1.1/1.2/4.1.1/4.1.2/6.4.5/6.5	44.5 10/15/10/2/7.5/5
Manmeet	-	-
Yahya	Total 5.1/5.2/6.4.5	17.5 10/5/2.5

March 2019		
Name	WP ID	# of hours
Chris	Total 2.0/3.0/4.1.3/4.2/6.5/6.5	65 3/2/5/20/15/20
Sean	Total 4.2/6.5/6.6	40 5/15/20
Manmeet	total/6.6	5/5
Yahya	total/6.6	5/5

April 2019		
Name	WP ID	# of hours
Chris	Total 2.0/4.2	24.5 4.5/20
Sean	Total 2.0/4.2	12.5 2.5/10
Manmeet	-	-
Yahya	Total 4.2	10 10

3.4 As-built Project Schedule

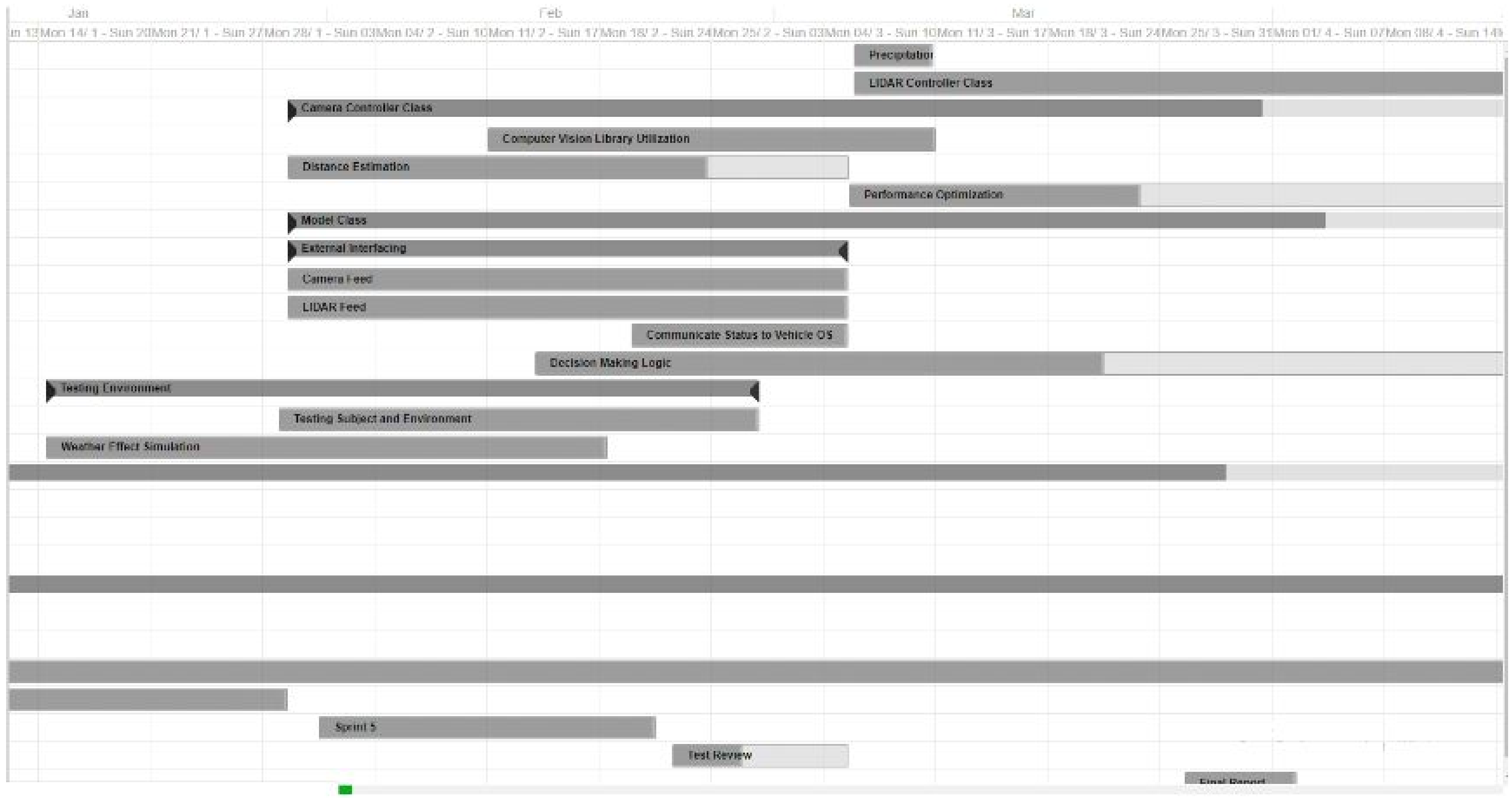

Figure 11 - Gantt Chart

3.5 Reflection on Management Strategies

The development team made use of the agile software development strategy to complete this project. Managed by a dynamically changing manager, weekly meetings were established with delegated tasks and bi-weekly reports were completed at the end of each sprint. At the end of the final sprint a test review was conducted on the system, although development continued on from that point. In this section the usefulness of this scheme will be discussed as well as it's application to the project itself.

Employing proper sprints early in the development proved to be a challenge for the team. With no direction and no clear team leader it was immediately apparent creating a direction for the project would be difficult. The team focused on independence of tasks rather than the impact of the tasks themselves, causing sprints to end with low advancements in the overall project. Especially early on, the sprints were spent researching the topic and exploring issues within the autonomous sensor calibration market rather than building a marketable prototype. In a big way with each sprint we were falling out of line with the agile manifesto, as a large aspect of the success of agile is the tangible work demonstrated and tested after a sprint has concluded. This failure was bred by a poor decision to choose the agile framework.

In hindsight, the agile framework was not suitable for this product. The calibration software was a small, precise software which was difficult to divide into sizable parts. One could not develop and test a small piece of the software without missing or relying on another piece. And while the system itself was split into separate classes, the weights of each class were nowhere near equivalent. For this reason sprints were either extremely busy, or extremely light.

The approach would have been a complete failure had it not been for our decision to create the testing environment to accompany our original product. The test environment is a suitable agile project. The api could be built piece by piece, tested and enhanced on a week to week basis. New features or designs could be introduced isolated from others, such that proper unit tests could be created and prove that the system was being properly maintained. Unfortunately for the group, this was not the project we were hoping to develop. However, the test environment allowed us to mirror an agile-like development plan over the course of the year.

With part of our team focused on the testing environment, part of our team developing the calibration software, and the rest focusing on proper documentation and execution, the team pulled through. While this didn't conform to the ideal agile environment, the team adapted the approach to our own needs.

3.6 Preliminary Business Case

3.6.1 LIDAR Sensor Calibration Software

As autonomous vehicles grow into a realistic purchase for a working class citizen, issues such as severe weather conditions must be explored in a cost efficient manner. The calibration software was developed as a direct response to the needs of the market, as major car manufacturers have expressed these issues. LIDAR systems struggle in a high precipitation environment, and this is unavoidable in many parts of North America. Figure 11 lists motivations and roadblocks for this system.

Figure 11 - SWOT for Sensor Calibration Software

Strengths - Low-cost labor - Minimal upkeep cost - Utilizes on-board systems - Compact solution in terms of storage - Multiplatform solution - Open-source	**Weaknesses** - Inexperienced team - Inability to test the solution on hardware - Relies on open-source libraries - Narrow scope of detection (solely license plates) which could be expanded - Relies on identifying objects, but has no clear solution otherwise
Opportunities - The best solution to this problem is apparently yet to be determined - Autonomous vehicle market is growing at a rapid pace - No cost to add into a pre-existing system	**Threats** - Competitors have teams of experienced engineers working on the problem - Companies may be reluctant to utilize product made by students - Companies may have privately developed better performing systems - Competitors are developing with greater access to resources

This product is specifically aimed at autonomous vehicle manufacturers who are looking for solutions to pre-existing problems without increasing the budgets of their already expensive vehicles. With the ability to be run on any platform with any array of sensors that should already exists on a modern autonomous vehicle, the calibration software is definitely worth exploring for anyone in the autonomous vehicle space.

The issues with this solution are mostly in the fact that it is unproven. In critical systems, it would be outright negligible to run this solution without extensively testing it in a real world environment. This drawback is something that can not be overlooked. Therefore, the price of operating the software would be to run tests on it, or otherwise obtain funding to properly test the software within the team.

Going forward, if the product proves itself reliable through test, its applications could be expanded to become reliable in areas outside its current scope. One glaring area of improvement is the algorithm's dependency on license plates, as it renders the system useless without the presence of surrounding vehicles. Other references, such as traffic lights or buildings could be used in a similar manner by triangulating object sizes and making comparisons. This would surely bring large appeal to the product.

3.6.2 Testing Environment

The market for the testing environment is derived from the exponential growth in autonomous automobile research in the past decade. While modern simulations cannot substitute the variability of real world testing, there is still demand for them outlined in Figure 12.

Figure 12 - SWOT for Testing Environment

Strengths - Virtually no operation cost - Limitless amount of tests can be run - No risk of accident - Ease of use	**Weaknesses** - Can never substitute real world testing - Arguably never truly random
Opportunities - Rapidly growing demand for this product - Current market options perceived as unreliable	**Threats** - Prospective users view this as a fruitless effort - Existing simulation environments fill similar niche

This product could be marketed to autonomous automobile manufacturers, or even third party software companies, who require cost efficient testing. The field requires an enormous amount of capital to enter, restricting many small firms from competing and discourage the reckless testing of preliminary software solutions.

No company in good faith could rely on a simulation to be there sole method for testing such a large scale critical system when the opportunity for real world testing is available. That being said, no company would knowingly test early development software on expensive vehicles and risk the destruction of precious resources. This product fills the intermediary testing situation between newly written software and an untested market ready product.

Going forward this project can expand in the variability of the simulation, bringing it as close to a realistic experience as possible. This could be through expanding the code base, API, or even utilising machine learning to imitate situations on real roads. This would only increase the confidence of these tests, putting a greater value into the results and therefore the value of the system.

Volume IV - Lessons Learned

4.1 Deviations From Plan

Our project underwent a significant deviation from the initial plan we had developed in the agile roadmap. The initial product that we aimed to build was a camera calibration system which would automatically adjust camera parameters in response to changing conditions based on the camera input and other sensor data. As we completed sprints with the initial topic, we could not generate user stories that could connect with the product we were researching and developing. This spurred our group to pivot from our initial idea of camera calibration, to LIDAR calibration. We decided to tackle a pressing problem in the LIDAR calibration field and one which we could tie to user stories. This pivot occurred in mid January, and although the new solution was focused on calibrating a different sensor, it still utilized the camera, which meant that some of the work from the first topic could be salvaged.

The next deviation occurred in mid February, and the choice we made was to remove some non-essential features from the minimum viable product. The decision to move these features to the backlog was motivated due to an inability to develop the features in the time that we had remaining. The rationale behind adding these features in the first place was that we felt that the product needed to be spectacular in order to justify switching so late into the development cycle. Narrowing our focus proved to be a necessity, so we opted to strip away non-essential components from the project to make the minimum viable product reachable within the scope of the course.

There were cases where we deviated from planned deadlines, but these were unfortunate effects of the difficulty of the project. Our team is still completing the minimum viable product as of the writing of this report, and after that we will perform a test delta before demoing at Lassonde Day. Naturally this is a deviation from the planned completion date, but our team has been struggling to allocate time to finish development over the past month. The plan going forward is to utilize the time between the end of exams and Lassonde Day, a two-week period where our team has no commitments other than completing the project. We are determined to present a working product to demonstrate that our effort up to this point have not been for nothing

4.2 Failure Report

Initially during the requirements elicitation, our group had been unable to meet/correspond with the customer, which increased the difficulty of the task significantly. Different members within the group also had different interpretations of the problem itself, which could not be resolved in time for the requirements review. This lead to a set of broad and inaccurate requirements which did not reflect the nature of the problem, which was pointed out to us at the requirements review. This was the first failure we encountered, and what we took away from it is that it is essential for all group members to be on the same page about the project. This failure sparked a discussion for our group, and from it we improved the way we conduct meetings by encouraging an environment where people can speak their minds and challenge one another's ideas without worrying about being wrong. A group that is on the same page, but has a misunderstanding of the problem, is stronger than a group where one member has a good understanding but the rest do not, because the group on the same page can reach an understanding of the problem and collaborate better.

Over the course of the early sprints, our team lacked an understanding of the agile methodology, and as a result we operated more like a waterfall development team than an agile one. Our early sprints consisted of research in the sensor calibration domain and acquiring the knowledge required to create computer vision programs. During these early sprints we did not utilize user stories, which delayed our realizations about the flaws of our first proposed product. The main thing that we learned from this failure is to connect the product to user stories as soon as possible. If our group had determined earlier that our initial product did not have a market fit, we could have pivoted much earlier and utilized all of that time. If we had done the agile methodology properly, than we probably would have finished our product in time for this final report.

An area in which our team failed was in meeting the later deliverable deadlines. Upon reaching the time to perform the test review, our project was not in a state in which we could do meaningful testing, and the same can be said of this final report. Our team failed to allocate enough labor hours to complete tasks by deadlines, and and it reflects poorly on us. The main takeaway from this failure is to be more determined to accomplish work in the times where there is less pressure. The least productive points of the projects development occurred at times when our group members an abundance of free time and deadlines were far away. This behavior pattern is probably common for students, but it is not reflective of professional engineers, which our group aspires to be.

4.3 Lessons Learned and Going Forward

The team has learned an immense amount about the agile development process as well as the project management process. Prior to this project, team members had some experience with agile development in industry, but now we are very familiar with it. We have learned that agile is a powerful and flexible development strategy which enables developers to connect what they are building directly to the various applications. At the same time, we have learned the limitations of agile development, in that it is not ideal for certain projects. For this project, we found ourselves developing towards a single product most of the time rather than to individual components which could have applications of their own. Having the hindsight available now, the team would likely have either elected to use waterfall development for this project, given the abundance of interrelated components. If we were to utilize agile development, it would be for a different topic than our current one.

The team did not run the agile development method smoothly for the first three sprints, but eventually executed it smoothly. Having the experience with user stories that we now possess, the team would have discovered the insufficiencies of the originally proposed product by the first sprint, and we would not have wasted as much time. The requirements elicitation process was extremely challenging for our team, and if we could repeat that process with our current experience, we would have pursued our sponsor harder for definitive requirements. The sponsor gave us autonomy in defining the requirements of the project, they did however restrict us to a specific domain. The team has learned from this project that requirements are the foundation of the engineering process, and leaving their definition in the hands of an inexperienced team like ours lead to significant difficulty fulfilling them.

With the hindsight now available, the team would have begun the testing process far earlier than was done for this project. The team was testing components of the project when the test review was due, and we had not yet begun building the solution out of those components yet. This reflected very poorly on our team, as it should have, and the disappointment from this has taught our team to begin testing much earlier than is required. Earlier testing would have made it clear how far we were from the minimum viable product, but our team was negligent of the amount of work that was required to move from testing components to testing the whole.

One of the most important lessons learned by the team from this project is the importance of leadership. Our team would have greatly benefitted if we had adopted the traditional team structure with an appointed leader to hold members accountable. The team has also learned to be more objective about our capabilities. On two occasions, the team had to reevaluate whether or not certain aspects of the projects could be achieved given the level of competence and availability of the team.

4.4 Evaluation of Problem Solving Process

This problem is one being tackled by teams with far greater resources filled with members whose ability and capacity exceed our own. The problem solving process we utilized was influenced by the depth of our understanding of the problem domain and our ability to learn difficult concepts. The flaws in this approach are immediately apparent, it would be optimal to learn everything there is to learn about the problem domain to truly learn whether a particular solution is wise. This is something we could have done better if we had chosen this topic from the beginning, as it was a challenge to delve deeper into the problem domain when we had so little time to prepare deliverables. A critique of our problem solving process could be that we wanted to solve the problem in way which utilized the time we spent in the camera calibration domain.

Another way in which our problem solving process could be improved is by consulting more with the advisors of the project. To our own admission, our experiences with our sponsor left us feeling disappointed with our support network, and this lead us to be more independent, possibly to our detriment. If we had consulted with experts in the sensor calibration field, we could have learned directly what challenges a team of students could potentially solve. Interacting with supervisors more would have benefited our problem solving process by learning how professionals operate and evaluate potential solutions.

A way which we could have improved our problem solving process would have been to press the sponsor harder for a defined problem. For this project, we were not given a problem to solve, we were given a domain in which to choose a problem to solve. The openness of this situation leaves countless problems to choose from and methods to solve them. This can be counterproductive to the problem solving process because there is always uncertainty whether the problem is difficult enough to justify it being the focus of the project. This can lead to overly complicated solutions when a more optimal and simplistic one is available, because the team feels the need to justify itself. In the case when the problem is clearly defined, a simple solution would be viewed as a positive, because clearly the problem had to be considered complex enough to be accepted as a capstone topic.

The way we approached solving this problem was from the beginning influenced by the disciplines that our group members are in. The approach we took was one that focused on the software side of the problem, when an approach emphasizing hardware and electronics could have been taken. This preference to work with programming tools shaped the way we approached the problem and the solution we came to. If we had brought on an electrical engineer to the team, then they might have brought a different perspective to the problem that could have lead to a better solution.

4.5 Self-evaluation

Figure 13 - Self Evaluation

Criterion	**Self-Evaluation Ranking**	**Justification**
Explain the importance of compliance with the Professional Engineers Acts and other relevant laws, regulations, intellectual property guidelines and contractual obligations and follow best practices	Meeting	See Volume V on PEO Acts and relevant information
Employ strategies for reflection, assessment and self-assessment of team goals and activities in multidisciplinary settings	Exceeding	There are numerous sections of the report dedicated to reflecting on our process and our results. The team assessed itself an objective and critical view in multiple sections. The self-reflection included difficult conversations like lessons taken away from failures and how the team would have done things differently.
Adhere to written instructions in a professional context	Exceeding	The report contains everything one would expect to see in a professional report. The technical volumes cover all of the required material, and we have created a separate technical volume for the testing environment. The management sections of the report are very detailed, and they include sections not listed in the outline of the report.
Evaluate critical information in reports and design documents	Meeting	Detailed design critiques can be found in relevant technical volumes with detailed information regarding the performance decisions and tradeoffs made.
Appraise possible improvements in the problem solving process	Exceeding	Well documented plans for future development. Acknowledgement of past failures and recalling of lessons learned from said mistakes. Detailed

		breakdown of management strategies and improvements that can be made. The team analyzed how it could have approached the problem differently.
Justify the strength and limitations of the solution and make recommendation for possible improvements	Exceeding	Well documented plans for future development. Paints a clear picture of what the product aims to be, what niche it can fulfill in a marketplace, and how it can improve to fit that niche. Good marketable strategy and plans to increase said appeal. Strong understanding of where products lack or will not be able to excel.

Volume V - Exclusionary Clause in Accordance PEO Act Standards

The AMD2 Autonomous Vehicle LIDAR Sensor Calibration System has not been tested in a real world setting. The system has experienced limited testing in a simulated environment. Subsystem tests are limited and prone to manipulation. Foundational libraries of these technologies have not been developed by engineers from this project and therefore cannot be guaranteed to be fail safe.

The AMD2 Autonomous Vehicle Simulation Environment is in no way an accurate representation of the complications of real world testing. Foundational libraries used in the development of this system were not intended for professional use. These technologies have not been developed by engineers from this project and guaranteed to be fail safe.

The products described in the above document are unfit for use in any critical system defined as follows: a system which must be highly reliable and retain this reliability as they evolve without incurring prohibitive costs. This includes but is not limited to: safety critical, mission critical, business critical and security critical systems.

By using these technologies in critical systems without obtaining permission by AMD2 engineers one may be subject to a professional misconduct. Use of this technology in a critical system would fall under PEO Act Section 72 (2) (a-e), and would violate Section 77 (1.i-iii) and (2.1).

This product does not have the seal of an AMD2 P.Eng and should be treated as such.

References

Samuel Markings - Sciencing. *A Tool Used to Measure How Much Rain Has Fallen*. 25 April 2017. https://sciencing.com/tool-used-measure-much-rain-fallen-22537.html. 02 April 2019.

Seegrid. *Seegrid - This company has developed stereo cameras for driverless cars*. 02 December 2016. https://www.cnbc.com/2016/12/02/seegrid-sensors-for-driverless-cars.html. 02 April 2019.

Voyage, Oliver Cameron -. *An Introduction to LIDAR: The Key Self-Driving Car Sensor*. 9 May 2017. https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff. 02 April 2019.