# User Augmentation Information

DESIGN DOCUMENT

**Team Number:** SDDEC19-14
**Client:** Mr. Radek Kornicki from Danfoss
**Adviser:** Aleksandar D
**Team Members:**
Omar Abbas - meeting scribe/report manager
Dennis Xu - chief engineer/meeting facilitator
Aaron Michael - chief architect
Jonah Bartz - continuous integration

**Team Email:** sddec19-14@iastate.edu
**Team Website:** http://sddec19-14.sd.ece.iastate.edu/

# Table of Contents

# List of figures/tables/symbols/definitions:

**List of Figures:**



**List of Tables:**



**List of Definitions:**

HUD - Heads up Display
GUI - Graphical User Interface
HMI - Human Machine Interface/Interaction
NFR - Non-functional requirement
FR - Functional Requirement
UART - Universal Asynchronous Receiver/Transmitter
USB - Universal Serial Bus
TobiiSDK - Software development kit used for the tobii eye tracker 4c
GazeSDK - Software development kit used for the Android devices use with the tobii eye tracker 4c
CPU - Central Processing Unit
GPU - Graphics Processing Unit
ARM Processor - An ARM processor is one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM) (Bigelow 1)
X86 processor - A CISC (complex instruction set computer) based computer architecture that was developed by Intel and AMD
ROS wrapper - Robotic operating system used for drivers
GPIO - General purpose Input Output
SSH - Secure shell protocol used for communicating with the shell of machine remotely

# 1 Introduction

## 1.1 Acknowledgement

We would like to first thank Mr. Radek Kornicki for providing supplies and guidance for the project. He has been providing us with all of the materials that we need to pursue this project. We also want to thank Dr. Aleksandar Dogandzic for supporting us with advice about how to develop our project.

In addition, the Virtual Reality Center on Iowa State's campus has provided support by showing their virtual environment. While augmentation is different from virtualization, there are many similarities between the two subjects, and the Virtual Reality Center helped by providing support about the eye tracking device.

## 1.2 Problem and Project Statement

There is a lot of danger when it comes to operating heavy machinery. These vehicles can do a lot of damage to people and property if not properly used. In many cases, the user has a hard time knowing what is happening around them. This, in conjunction with a distracted user, can provide a very dangerous environment to work in. Mr. Radek Kornicki from Danfoss wants to determine the viability of a user information augmentation system in improving safety when operating heavy machinery.

Mr. Kornicki wants to test how feasible a heads-up-display (HUD) system is to increase safety. The system will show visuals about the environment while the user is operating heavy machinery. To accomplish this, we want to make the windshield of the vehicle a HUD. This HUD will give information to the user about potentially dangerous situations that they might be in. This system will indicate things that the user should be looking at and also determine whether or not the user is doing something dangerous.

## 1.3 Operational Environment

Our client is the The Danfoss Group, known for manufacturing products and providing services used in powering heavy machinery, solar enginery refinement, food preservation and much more. The R&D division of Danfoss is determined to improve the technology in off-highway vehicles including agriculture and construction. Our embedded system may be used in any of these branches and given their agricultural focus we may safely assume that this can be put inside a vehicle that is moving in a muddy terrain or other less than ideal conditions. Hence the actual module itself should not be exposed but the cameras monitoring around the proximity of such a vehicle may in fact be affected. In our design the cameras will be separate and if this is the case, they should not be exposed to extreme heat, wet or other hazardous conditions for cameras. The module itself is 5 cm by 9 cm but should be kept stationary away from hazardous elements.

## 1.4 Intended Users and Uses

The intended users for our solution are heavy machinery operators. Heavy machinery operators are workers who using heavy machinery (like combine harvesters, cranes, and bulldozers). Since the workers operate heavy machinery for a job, we can't assume that the user has a lot of technical experience. This means that we need to design a system that is easy to set up and easy to use. The

images we draw onto the screen must also be clear and concise. This is because our product is targeting people in a busy work environment.

The intended use for our solution is improving safety when someone is operating heavy machinery. The solution needs to be fairly generic so it can be put into a lot of different machines without too much modification. That being said, there is a large diversity in heavy machinery with very different requirements. This means that we also need to make it easy to expand and iterate on the solution.

## 1.5 Assumptions and Limitations

As a response to conditions referenced to us by our client, we are to make the following assumptions:

- The module will not be used in a turbulent situation.
- Will be operated in an environment where power is provided.
- Will only have one user at a time.
- For the prototype, information will be projected onto a screen or windshield.

To justify these assumptions, turbulence may not be ideal in the situation where we are parsing video input as it may be shaky. We are also making the assumption that this module will be operated in an environment where the module will be connected to a power source. A battery may be installed later but for the purposes of project, the prototype is meant for stationary means and not mobile. Finally, the eye tracker will be calibrated for one person at a time, and as such, we are assuming that is the maximum number of users.

Below is a list of the limitations we are under:

- Tests on the module will be undergone in a stationary, rather than mobile environment
- Less support as x86 - Arm conflict is not commonly addressed online
- Will only have one user at a time
- Data projection must be 70% transparent as stated under Iowa windshield laws

In addition, since we are working on a college campus, we are limited to testing the system in a stationary condition. Until we have permitted to use the system in a mobile situation provided on or off campus, we will not be able to test its efficiency on the move.

## 1.6 Expected End Product and Deliverables

Our expected end product is to design and build an eye triggered interface that will assist in using heavy machinery and electronic applications. This interface will exchange information with the user regarding the environment they are in to give them better intuition with their technology and to tighten the grip between the user - machine interface. We are to deliver a prototype consisting of the aforementioned components that accomplishes the description as efficiently as possible.

Our final deliverable will be the system composed of the following items:

Hardware:

- Tobii Eye tracker
- NVidia Jetson TX2
- RGB and infrared cameras
- Projector

Software:

- Programmable framework to suit needs of the project incorporating Tobii/Gaze SDK
- Working lane detection and basic object detection of road signs
- Code for detecting if sensors are working
- Code for outputting frames for a HUD

# 2. Specifications and Analysis

## 2.1 Proposed Design

Our solution makes use of the hardware the client initially provided. In order to make an HMI (Human Machine Interface) system that interacts with human eyesight. The team is using the Tobii Eye Tracker 4C and a combination of normal cameras and infrared cameras for our object detection system. In this case, object detection is provided by a Python/C++ based image analysis library called OpenCV. For data projection, we will use projection film that is compliant with Iowa Department of Transportation standards and laws since that is the area where this project will be demonstrated. Voltage input required to power the module is 5.5 V at minimum and will be through means of an electrical outlet for the time being. The entire system is tied together on the Nvidia Jetson TX2, an ARM based platform.

We currently have the Nvidia Jetson TX2 up and functional with ubuntu. On the Jetson TX2, we also have corner detection working with a webcam and OpenCV. We tried getting a Tobii Eye Tracker 4C to work in this system but ran into issues, namely that we have object code compiled for an x86 system while we are using an ARM system.

This issue is where our different designs come in. One solution is to try to get the Tobii eye tracker working natively on the Jetson TX2. We can do this by either getting an object file from Tobii or via virtualization/emulation of a x86 system. We can go through a microcontroller that has an x86 architecture such as the Intel Quark Microcontroller.

Functional requirements of the module are:

- Can we identify lanes on the road
- Can we identify important objects on the road(signs, people)
- Can we track user's eyes in the vehicle
- Can we project this in a heads up display style
- A system that will track eyesight over a large visual field
- Can determine whether a sensor has been compromised

Non-functional requirements are:

- Object detection and eye tracking to be done simultaneously at real time
- Object detection will to be done with an acceptable level of accuracy
- Fast, reliable and clear HUD capable of keeping up with user while they do the task at hand
- Reliable and secure module protected from malicious users and hackers.
- A user-friendly calibration session for each user profile as pupils may differ due to different socioeconomic traits.

In order to achieve these requirements, we need to follow the Iowa Windshield obstruction laws.

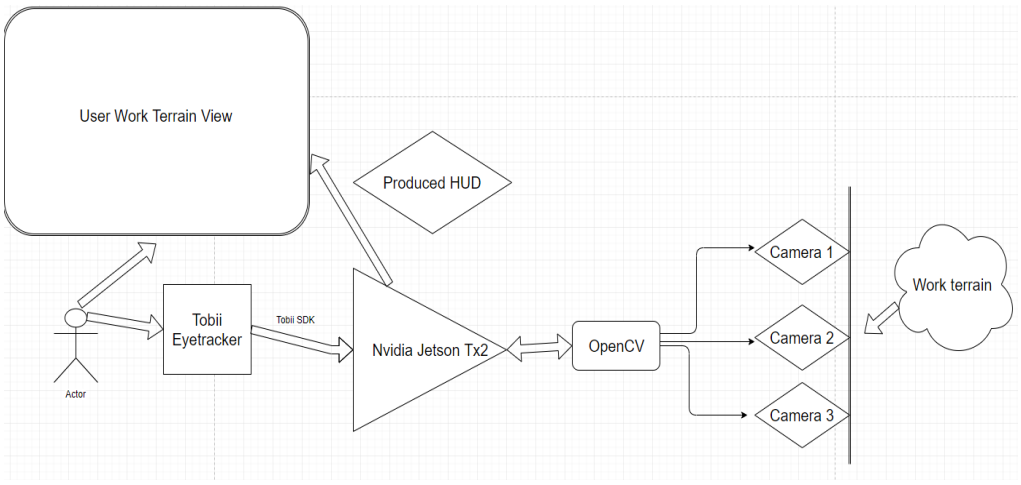


Figure 2: Hardware Design with Eye Tracker Connected to Jetson TX2

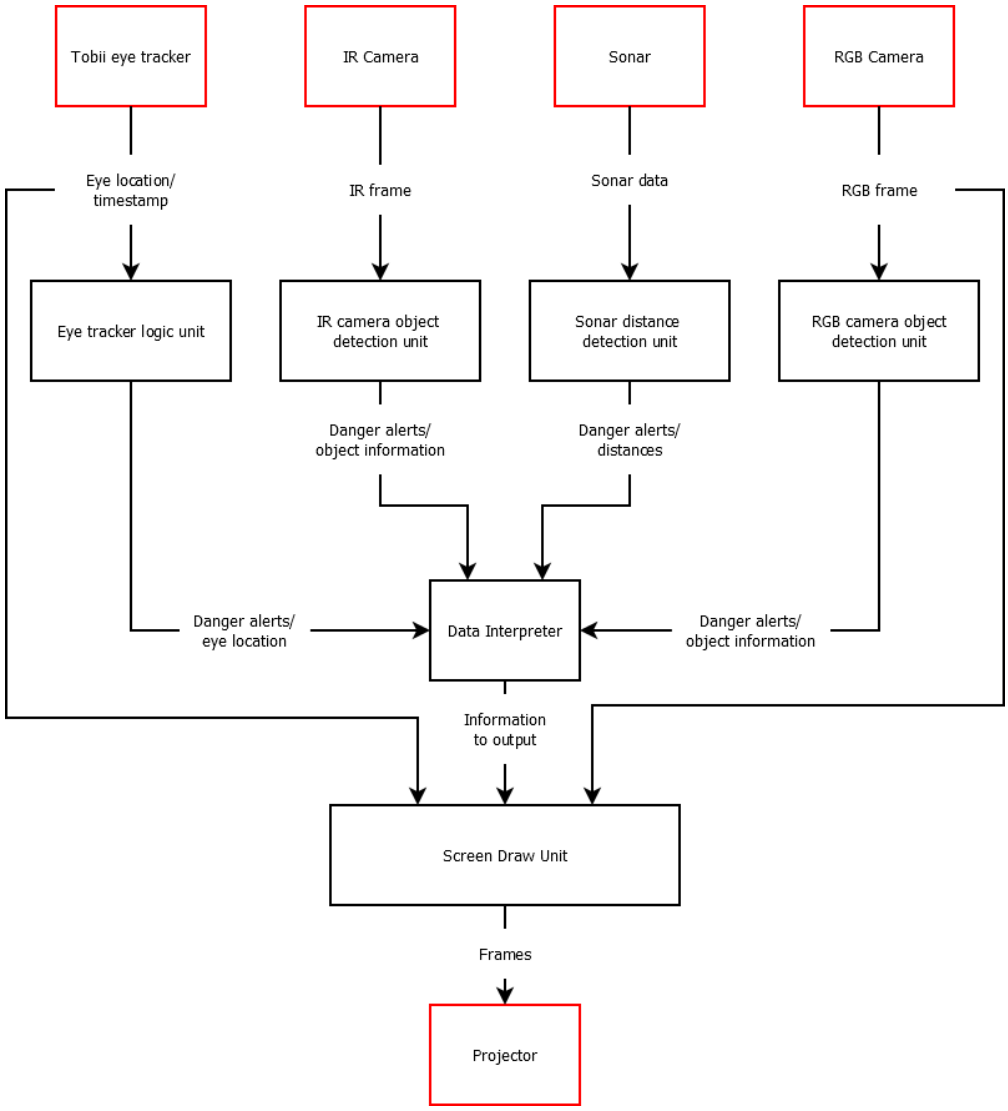


Figure 3: Software Design Diagram

## 2.2 Design Analysis

The analysis of our design starts with the requirements given to us by the client. We are to use an eye tracker to communicate with some module that will queue relevant information to a display. Therefore, we've come up with a potential architecture that obeys the limitations imposed on us as seen in *1.5 Assumptions and Limitations.* Figure 2 shows one of our hardware designs and figure 3 shows the software design.

In the proposed design, the Tobii eye tracker takes input from the user's eyes. This information is transferred to the Nvidia Jetson TX2 which is monitoring the work terrain at the same time. It is important to note that the position the user is looking at in the User Work Terrain View is the position the Tobii eye tracker will communicate to the Jetson TX2, as the purpose is to monitor whether the user is keeping track of certain things in the designated User Work Terrain View. The Nvidia Jetson TX2 would be taking input from 3 cameras and parsing the video input frame by frame using OpenCV to detect objects that are relevant to the user's purpose. This imaging is then returned by the Nvidia Jetson TX2 to the User Work Terrain view so it can simulate a heads-up display (HUD) to notify the user of certain things. This defines our flow for the system. However, these arrows merely represent information casted from one device to another. We are not taking into consideration how they communicate. Tobii Eye tracking uses software (TobiiSDK) that only supports x86 architectures, whereas our Nvidia Jetson uses an ARM based processor. Therefore, the drivers that Tobii needs at some of the lowest levels of operation will not be understood by the Nvidia Jetson Tx2 and thus will not be installable there. It will simply not work on the Nvidia Jetson TX2. Since it has not been determined yet whether we can directly connect to the Tobii Eye Tracker to the Jetson, we have decided to utilize different approaches:

1) Introduce a mediary between the eye tracker and the Jetson
2) Use an ROS Wrapper to convert x86 instructions to ARM instructions
3) Swap out the Tobii eye tracker for one that has ARM based processor drivers readily available
4) Use Gaze SDK rather than Tobii SDK

By introducing a mediary, we mean introducing a microcontroller with x86 architecture that can communicate with the Tobii eye tracker in the stead of the Jetson. It will then transfer the information from the user's eyes to the Jetson via the conventional UART (Universal Asynchronous Receive & Transmit) device. Figure 4 shows this design.

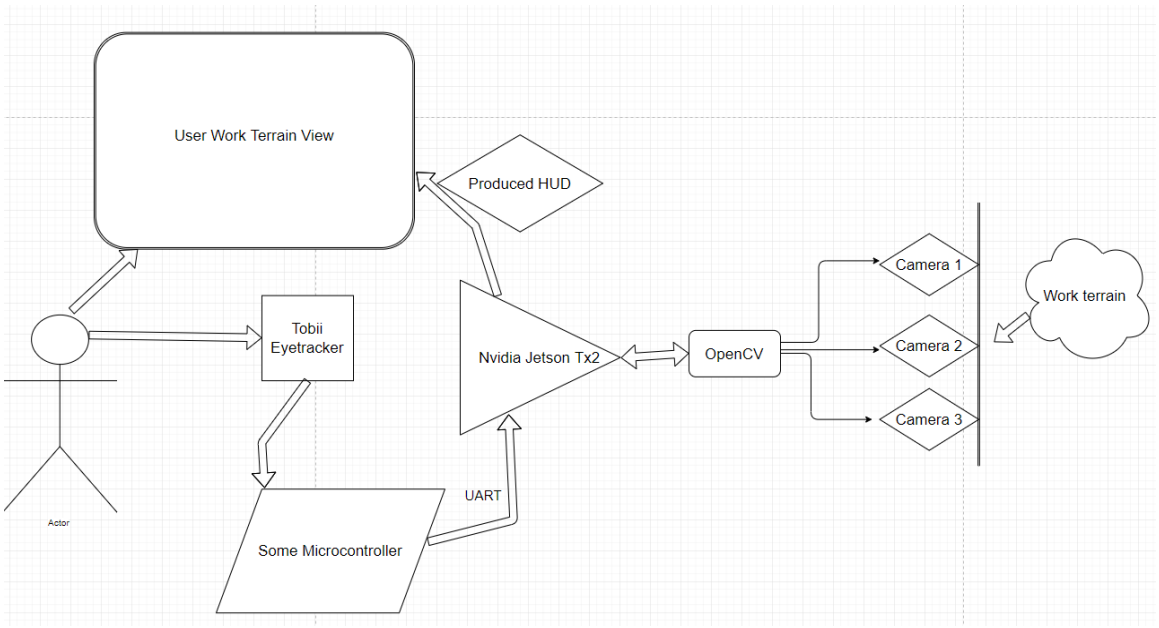


Figure 4: Hardware Design with Mediary Microcontroller

Later in our research, it has been revealed that we should look into ROS wrappers to drive the eye tracker on ARM. This can be represented as a straight arrow on our diagram as a potential solution alongside using the Tobii Gaze SDK software. Now, we will have to determine what cameras we should use. Various use cases specific to our client may include muddy situations or other agricultural environments that may be in the dark or light. Hence, more than one type of camera must be utilized for the purpose of object detection. So, we replace Cameras 1, 2, and 3 with RGB, IR and depth camera respectively. RGB is the regular camera view we are used to. IR camera is to detect edges of objects, and a depth camera may be used to measure the distance. This depth camera may actually be substituted with different sensors such as sonar. These sensors will also have code that will check the validity of the data. If the data is deemed wrong, we will indicate the user of the potential issue with the sensor.

Finally, we introduce what we will use to make our HUD. Our HUD will be displayed wherever the user should be looking in regard to their occupation. For example, if they are driving a vehicle then it will be the windshield. How we hope to achieve this is by using a projector to project the information onto transparent projection film that obeys standards set out by the Iowa Department of Transportation. This leaves us with the diagram shown in figure 5. This figure accounts for using either a mediary microcontroller or connecting the eye tracker directly to the Jetson TX2.

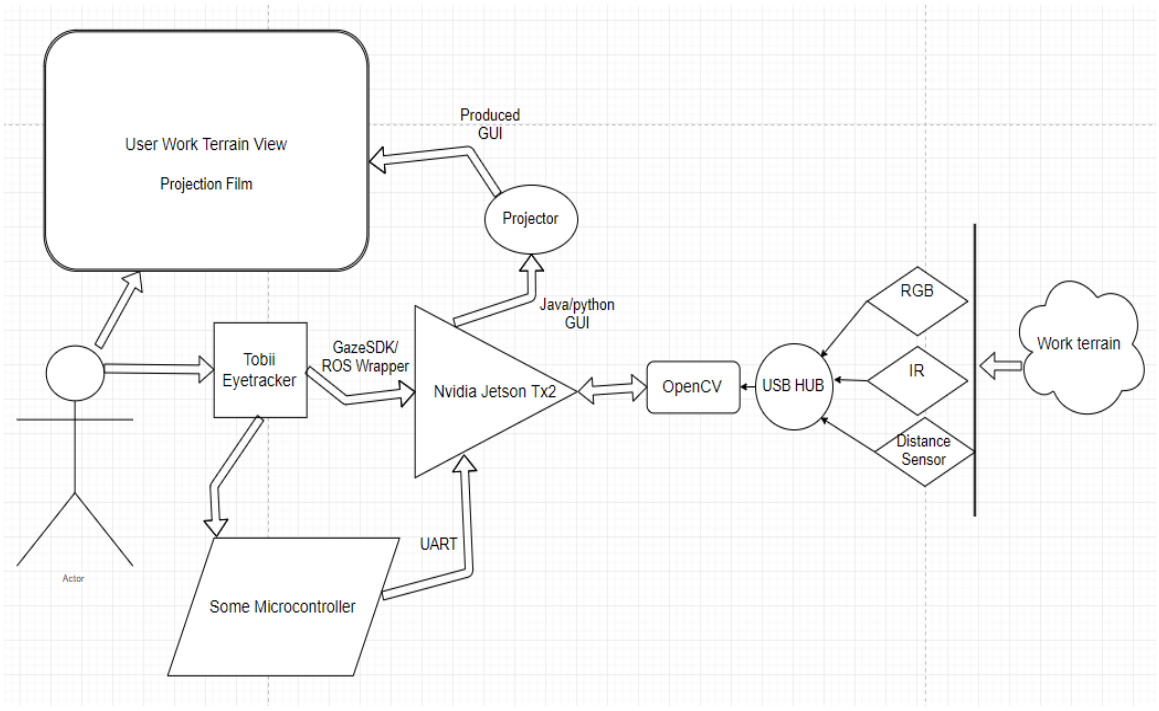


Figure 5: Detailed Project Design

This prototype is the best way to achieve this purpose. On one hand, there is plenty of hardware where the work needed to be done will be divided. The only multitasking device here would be the Nvidia, as it will be running a kernel with the required minimum dependencies, taking input from the cameras and parsing them frame by frame in search of scripted objects. This is where we believe a bottleneck may arise. The Nvidia Jetson TX2 is rated to be the best for this purpose as it has been utilized for a gaze tracking drone (Yuan 1). However, for our purposes the Jetson is parsing video input from 2 different channel inputs that are connecting via Universal Serial Bus port. This may cause traffic data and may be a potential weakness in our architecture

# 3 Testing and Implementation

## 3.1 Interface Specifications

The hardware interfaces that we are working on are dependent on the components that we receive. Ultimately most of the hardware will connect through a USB interface. The drivers will either be the default Ubuntu drivers or the drivers from the component's manufacturer. The only driver we will get use that is from the manufacturer is the eye tracker. If we require a secondary microprocessor, we will connect it through a the GPIO (General purpose Input Output) pins using UART with a connection speed is 115200, with 8 bits, no parity, and 1 stop bit. This will transmit the where the eye is looking at and the timestamp of the measurement. Table 1 shows the format the data will be in.

| Bit | Name | Description |
|---|---|---|
| 95-64 | timestamp | The time that the eye coordinates were taken. |
| 63-32 | y-location | The y positional location of where the user is looking. |
| 31-0 | x-location | The x positional location of where the user is looking. |

Table 1: Eye Tracker Data Format

Figure 2, the software design diagram, also includes the interfaces between the different software components. The components that are outlined in red are components where we don't have full control over. The components in black are components we need to create. The Tobii eye tracker, IR camera, RGB camera, and ultrasonic sensor (sonar) all transmit raw sensor data and send the data to their respective logic units. The eye tracker and the rgb camera also sends the same data to the screen draw unit. The logic units then send an alert about potential issues to the data interpreter. The data interpreter sends a notification to the screen draw unit to draw new things. The screen draw unit then output standard frames to the projector. This system allows us to test each component separately and also redo tests with data we collected before.

## 3.2 Hardware and Software

There is a lot of different pieces of hardware used in this project.

1.) Nvidia Jetson TX2
2.) Mouse and keyboard
3.) Normal RGB Camera
4.) Infrared Camera
5.) Eye tracker
6.) Projector
7.) External storage
8.) Valgrind

9.) GNU Debugger

**Nvidia Jetson TX2:** This is a module that will act as our main computing device. It has a GPU, CPU, and memory. It will be central unit that controls the rest of the hardware. We have this because it is specifically designed for image processing and has a strong GPU.

**Mouse and Keyboard:** This is trivial; but used to control the nvidia and develop on it. Ideally once the system is finished it will run our program on startup, but for developing and using it we need input devices.

**RGB Camera:** This will be used for object detection in most cases. We will want to detect lanes in a road, signs, and people.

**Infrared Camera:** This will be used for object detection when the "normal" camera isn't good enough. For example, at night it might be hard to see people on the side of the road so an infrared camera can capture the heat from the person and detect it.

**Eye tracker:** This will be used to capture a few pieces of information from the user. One is for detecting if the user is looking out the front windshield or not. We want the users to be looking in front and paying attention while operating the vehicle. Another thing we can detect is if the user's eyes are open. It is common for users to fall asleep behind the wheel, especially if they have been working in the field for a long time. This can detect drowsiness and alert the driver.

**Projector:** This will be used to display all the information gathered from the above hardware and display it on the windshield. It will create the heads up display and show important information without distorting the users view from the road.

**External Storage:** This will store data that we can run through the system during testing. The Jetson only has 32GB of on-board storage (1). Since we have multiple cameras recording, we might need external storage to store the different runs so we can repeat any situation.

**Valgrind:** This program lets us check for memory leaks in our C/C++ code. This is very important for us to use given bugs might cause the entire system to crash or even mislead the user.

**GNU Debugger:** This tool lets us step through C/C++ code. This will help speed up our debugging of C/C++ code.

## 3.3 Functional Testing

For testing we will be using a combination of software and hardware. We are testing to make sure what we are trying to do will be feasible. This will be before the actual software development has begun and will be a mixture of unit and system testing. We want to check if the hardware we have will be able to connect to the Nvidia Jetson TX2. This is important because the Nvidia is the main computer for our whole system. We have cameras, keyboards, mouse, eye trackers, and projectors that we are connecting all to the Nvidia. This will require testing to make sure that the functionality of each piece of hardware will be as expected when connected all together. We will test each piece of hardware individually and make sure that they work first, and then test if we can control them through the nvidia.

The second form of testing is to test our projected to our specifications. Ideally, we will be doing this type of testing after we have confirmed our hardware will be able to connect and work within the design specifications of our plan. This will also focus more on testing the software of our

system. This will be more of a traditional testing and will involve a lot of unit testing and then some system testing at the end. Unit testing will be most beneficial because we have so many parts that we want to make sure each part works individually. This will allow for not only simpler tests, but also quicker testing. For example, if we are working on just the eye tracker functions, we don't need to test the cameras. By splitting up the tests into individual tests will result in a more reliable system. After we are confident that each piece will work individually, we have to make sure that they can talk to each other. The unit test will be great for confirming if one piece works or not, but at the end we need the whole system working in unison. This will require some system testing. Ideally, we want to be confident in our unit test because system testing can be very difficult at times and a lot harder to debug issues.

There are a few functional things we want to test:

1.) Can we identify lanes on the road
2.) Can we identify important objects on the road(signs, people)
3.) Can we track user's eyes in the vehicle
4.) Can we project this in a heads up display style
5.) A system that will track eyesight over a large visual field
6.) Can determine whether a sensor has been compromised

In order to test lane/object detection we use pre-recorded videos we took and run them through the object detection. We can then confirm by hand to make sure that it's working properly. We can do a similar thing with the eye tracker. In order to test that the screen draw unit is drawing the proper information we can send it recorded data and alerts to draw and check the output. Testing that the system can track eyesight over a large visual field can be done by simply hooking up a Tobii eye tracker check the minimum and maximum ranges.

### 3.4 Non-Functional Testing

Non-functional things we want to test:

1.) Object detection and eye tracking done simultaneously at real time
2.) Object detection will be done with an acceptable level of accuracy
3.) Fast, reliable HUD capable of keeping up with user doing task at hand
4.) Reliable and secure module protected from malicious users and hackers.

Non-functional requirements (NFR) will only be tested after our functional requirements(FR) pass all the tests. It is not important to worry about NFR before we are certain the FR are working. We would rather have a functioning system then a pretty system.

To test the NFR we will use a similar method as FR. Unit testing will be important to test individual features.

For real time testing we will conduct a live demo with the system hooked up to a demo vehicle. We were told there might be a possibility to use a small vehicle at the Danfoss Application Development Center (a large track used for testing vehicles) here in Ames. If we cannot use this then we will import a live feed from footage taken from a car on normal roads.

To test the accuracy of object detection we have decided that an 90% success rate will be passable. To test this, we will conduct a unit test which will display 10 objects and our system should correctly identify 9.

Heads Up Display testing will be more of a system test. The heads-up display is in charge of displaying the information gathered from the system so we will want to save this until the other parts are passing their tests. To test this will be to run the system as a whole. We want to see if the HUD can project the information gathered.

Finally, for security we will have to try and break into our own system. We don't want a hacker or malicious user to affect the information displayed on the heads-up display. To combat this, we will run our system as administrator and test different attack methods like SSH or desktop share to see if we can crash the system. If we can we know we will have to change the way it runs.

## 3.5 Process

To test the methods described in Section 2 of this design document unit tests will be used. These will allow for a swath of standardized tests to be implemented during the course of production to ensure that all aspects of the project work together and adhere to the method, including functional and non-functional requirements. One way to test the project is using a predetermined video input. The product will be tested using a sample loop of video with objects and input data such as speed, tire pressure, etc. that will be consistent throughout the production of this project. The video shall be taken from the vantage point that the real-world usage will take place. It will include road signs, obstacles, and lanes on the road. This will allow for a baseline to test against as the team will establish which object types shall be detected in the video and will be able to match if the application detects these. Additionally, it will enable us to display the information from the vehicle onto the windshield. Because there will be a prerecorded video, we can test cases where the driver does not look at the alert, looks away for too long, or a range of activities that are not safe for the driver to be engaging in while driving heavy machinery.

Below is the flow diagram for the project. A feature of the project will be produced from the product backlog and marked as needing testing. The feature will be subjected to a series of unit tests to ensure that the code performs as it should. Then it will be evaluated by the team to ensure that all functional and non-functional requirements have been satisfied. If it has the feature will be marked as completed. If not, identify the requirements not met. The team will brainstorm ways to satisfy these requirements and the feature will be updated and bugs fixed. This loops until such a time that the requirements are met, and the unit tests are passed.

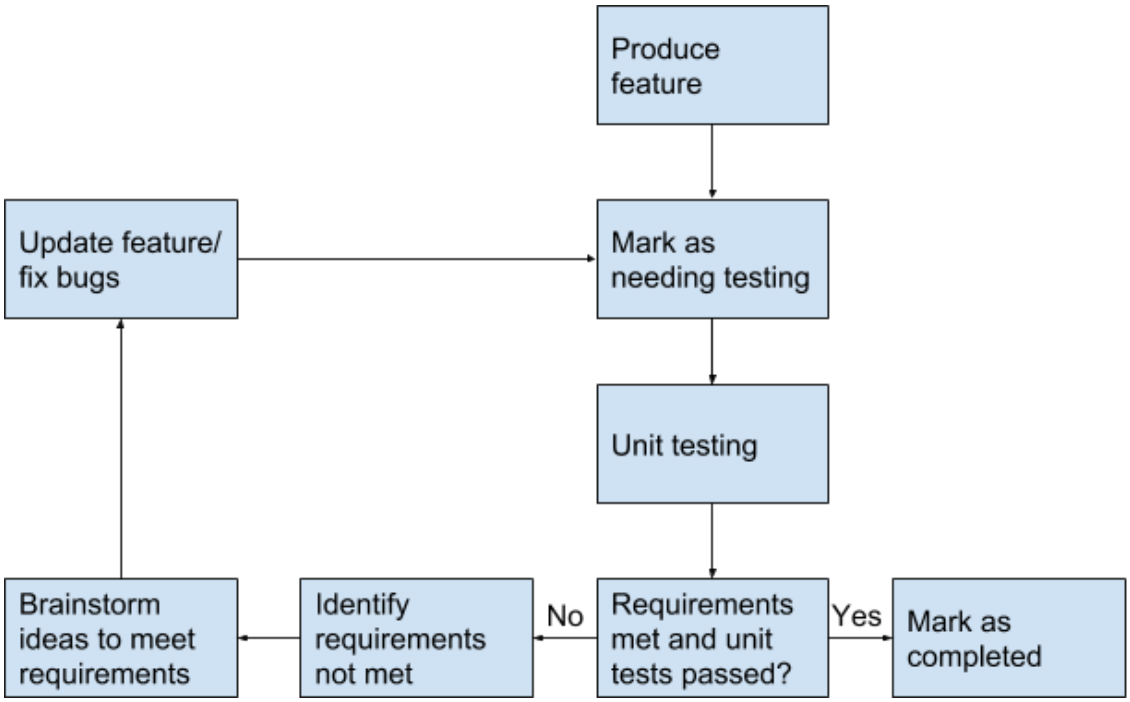


Figure 6: Flow Diagram

## 3.6 Results

Thus far in the lifetime of the project the team has been testing the system through connecting each part of the system together. At this point, we have had success in flashing the Linux environment onto the Jetson TX2 and are able to use the Jetson through the use of a monitor, rather than a projector, as it is still to be determined which projector will fill the needs of the project. The team has had success with corner detection using the camera on the Jetson TX2. Lastly, the team has had failure when attempting to connect the Tobii Eye Tracker 4C to the Jetson TX2. The Jetson uses an ARM processor while the SDK for the eye tracker is for an x86 system. This could be a potential blocker to development of the project going forward. One possible solution the team is looking at currently would use an ROS wrapper to address this issue.

Implementation Issues and Challenges:

1.) The Jetson TX2 uses an ARM processor while the Tobii Eye Tracker 4C SDK utilizes an x86 processor compatible file, creating implementation issues.
2.) Getting everything working on the Jetson TX2 without any noticeable slowdown.
3.) Determining whether or not a sensor has been compromised.

# 4 Closing Material

## 4.1 Conclusion

We currently have some basic components working on the Jetson TX2. With the notable exception of the Tobii eye tracker. Ultimately, we want to design an economically viable HUD system that will indicate to the user about potential dangers when they are operating power equipment. To do this we are using RGB and IR cameras to detect lanes and objects in-front of the vehicle such as important signs or pedestrians. We are also using a Tobii eye tracker to determine where the user is looking and see if there is an issue with attention. These devices are then connected to a Jetson TX2 to do processing.

The design is a careful balance of the improvements and the cost of the solution. The windshield display saves on cost and some complexity. The eye tracker we are using also helps due to the fact that it's well supported by the developers as well as the open source community. The object detection is also it the realm of being doable in two semesters. Ultimately, we feel confident in our current design given that we get the eye tracker working. If not, we also have a backup plan.

## 4.2 References


1. NVIDIA. (2019). *NVIDIA Jetson Modules and Developer Kits for Embedded Systems Development*. [online] Available at: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/ [Accessed 27 Mar. 2019].

2. iowadot.gov. (2019). *Iowa Window Tinting Standards*. [online] Available at: https://iowadot.gov/mvd/resources/windowtintingstandards.pdf [Accessed 27 Mar. 2019].

3. Yuan, L., Reardon, C., Warnell, G. and Loianno, G. (2019). Human Gaze-Driven Spatial Tasking of an Autonomous MAV. *IEEE Robotics and Automation Letters*, 4(2), pp.1343-1350.

4. Virtual Reality Applications Center. (2019). *VRAC*. [online] Available at: http://www.vrac.iastate.edu/ [Accessed 27 Mar. 2019].

5. WhatIs.com. (2019). What is ARM processor? - Definition from WhatIs.com. [online] Available at: https://whatis.techtarget.com/definition/ARM-processor [Accessed 27 Mar. 2019].

6. Tobii Developer Zone. (2019). Getting Started - Tobii Developer Zone. [online] Available at: https://developer.tobii.com/consumer-eye-trackers/stream-engine/getting-started/ [Accessed 27 Mar. 2019].