CarMa

Authors: Jananni Rathnagiri, Adriana Martinez, Evann Wu: Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—CarMa is a driving assistant tool designed to guide drivers to follow safe protocols. It would primarily be used to help new and inexperienced drivers get comfortable on the road. Our driving assistant would simulate the role of a parent or driving instructor in the front seat warning the driver about distracted driving. We plan on having a camera pointed at the driver and using computer vision to detect if the driver is falling asleep. This would prevent drivers from falling asleep at the wheel.**
CarMa primarily focus on ensuring the driver is focused on the road using computer vision to track the driver's face and movements. The domains used in this project are Software along with Signals and systems. The driving scope is focused on residential roads and our project requires there to be adequate lighting on the user's face.

***Index Terms*—Driving assistant, Computer Vision, Facial Detection, Facial Landmarks, Eye tracker, Mouth Tracker, Embedded System**

1 INTRODUCTION

Distracted driving is the cause of every 1 in 5 deaths from vehicle related accidents [2]. It is a issue rampant among young drivers learning how to drive for the first time. We believe it is critical that drivers learn safe driving habits as they are learning to drive. Our project CarMa is an automated driving assistant that takes real time video data of the driver and alerts the driver when they are becoming drowsy or looking away from the road for too long. While driver monitoring is a well explored field, especially monitoring drivers in the trucking industry, these applications are not widely available. Our approach to this problem is to design a system that is both compact and computationally powerful giving good performance as well as being easy to install.

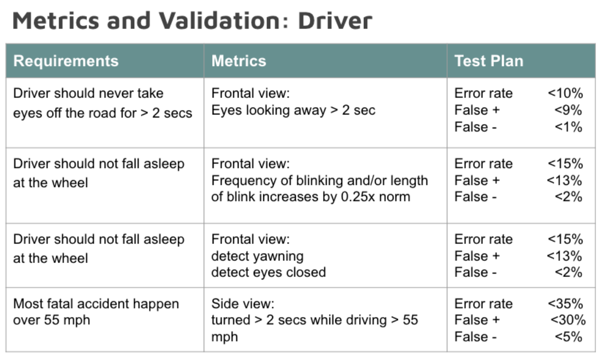
Our goal is to have our application run at least 5 frames per second and to achieve a 75% accuracy on our test suite. These goals will allow the application to be performant and limit the false negative rate. Our key metrics, frames per second and accuracy, were chosen specifically to ensure that our application is running efficiently enough to support our desired goals.

2 DESIGN REQUIREMENTS

The main requirements for this project include that the driver should never take their eyes off the road for over 2 seconds. This is taken from the National Highway Traffic Safety Administration or the NHTSA. Another requirement is that the driver should not fall asleep at the wheel which is essential to the project goals. According to the NHTSA, most fatal accidents occur when a vehicle is going over 55mph. So, CarMa should guarantee driver monitoring over that speed.

Given these requirements, we put together metrics and test plans in order to ensure the project meets expectations. The metrics have been split up into 2 main categories: Driver Metrics and Device Metrics.

Here are two tables that fully illustrate the requirements, metrics and testing plans.



Metrics and Validation: Driver

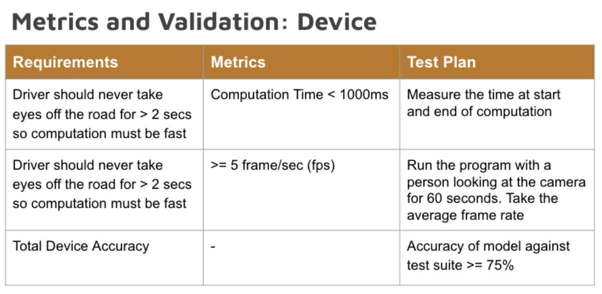
Requirements	Metrics	Test Plan
Driver should never take eyes off the road for > 2 secs	Frontal view: Eyes looking away > 2 sec	Error rate <10% False + <9% False - <1%
Driver should not fall asleep at the wheel	Frontal view: Frequency of blinking and/or length of blink increases by 0.25x norm	Error rate <15% False + <13% False - <2%
Driver should not fall asleep at the wheel	Frontal view: detect yawning detect eyes closed	Error rate <15% False + <13% False - <2%
Most fatal accident happen over 55 mph	Side view: turned > 2 secs while driving > 55 mph	Error rate <35% False + <30% False - <5%

Figure 1: A table of the Driver Metrics

Looking at the driver, there are two main positions the driver can be in: directly facing the front (frontal view) and turning away from the camera (side view). For frontal view, there are a few characteristics CarMa should identify:

- Eyes looking away for over 2 seconds
- Higher frequency of blinking
- Longer blinks
- Yawning
- Eyes closed

For these metrics, we decided that it is better for us to get false positives rather than false negatives. In other words, we prefer if CarMa classifies the driver as distracted or sleepy and warns the driver when they are in fact attentive. But on the flip side, if CarMa fails to identify a distracted or sleepy driver this could have fatal consequences.



Metrics and Validation: Device

Requirements	Metrics	Test Plan
Driver should never take eyes off the road for > 2 secs so computation must be fast	Computation Time < 1000ms	Measure the time at start and end of computation
Driver should never take eyes off the road for > 2 secs so computation must be fast	>= 5 frame/sec (fps)	Run the program with a person looking at the camera for 60 seconds. Take the average frame rate
Total Device Accuracy	-	Accuracy of model against test suite >= 75%

Figure 2: A table of the Device Metrics

For the device, based on the fact that the driver should be warned if their eyes are off the road for 2 seconds, the computation must be fast. Therefore, we estimate our round trip computation time must be under 1 second. Also based on research for the algorithms and our board, we would like to see a frame rate of under 5 frames per second. Finally, for the overall accuracy of the device, based on the models CarMa uses, the accuracy of the device against the test suite should be over 75%.

3 ARCHITECTURE OVERVIEW

The overall architecture will loosely follow the diagram in Figure 3. The diagram demonstrates how the device will be positioned in the car along with how the components will be attached.

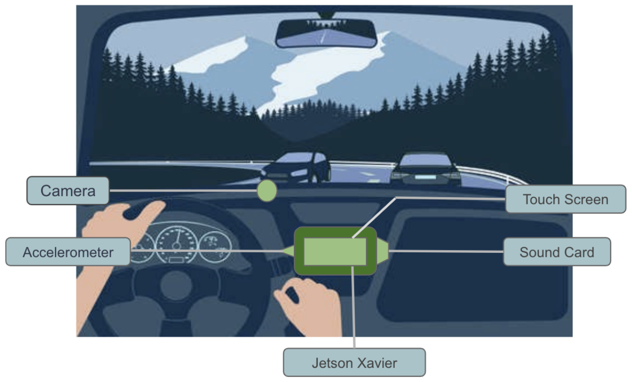


Figure 3: Mock-up of our Device

3.1 Hardware

The hardware that CarMa needs is split into two main parts. First, there is the Nvidia Jetson Xavier NX board. This boards has a powerful processor and GPU that allows it to output up to 21 Trillion Tera Operations per Second (TOPS). We believe with this compute power, we will be able to achieve the desired frames per second using our algorithm based off previous benchmarks set on similar hardware. The Jetson board utilizes the Ubuntu operating system which allows us to use Jetpack SDK which contains common CV libraries optimized for the Xavier board. Inside Ubuntu is where we will have all our software algorithms which we will discuss in the upcoming sections.

Second, we have the accessory hardware components to connect to the Xavier NX board. This includes the camera, touch screen, sound card, and accelerometer. The Sound Card will output warnings back to the user when they appear distracted. The accelerometer will be used to ensure that we are adhering to safety measures when the driver is driving fast or completely still. The Camera will serve as the input to the computer vision aspect of the project. Lastly, we have our touch screen which will act as the initial configuration and user interface. This will be how the user interacts with CarMa. These components will allow us to sample data while the user is driving which will be used as input to our application. The block diagram in figure 4 outlines how the hardware for this project will be connected.

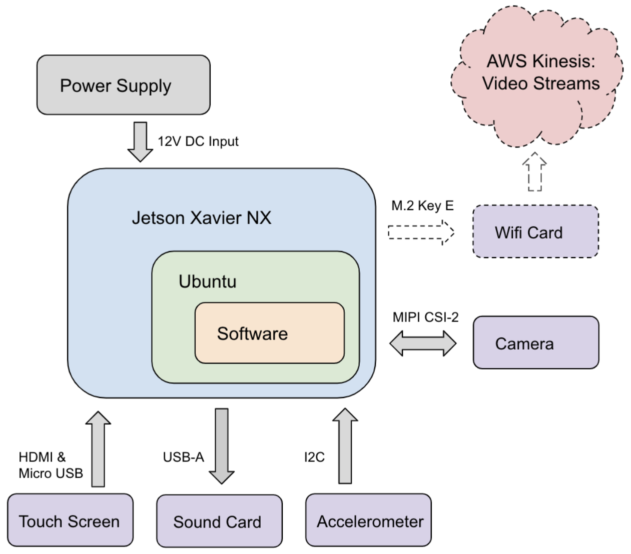


Figure 4: Hardware architecture of CarMa

The diagram in figure 4 contains a dotted path which is an optional route that we might pursue. In case the Nvidia Jetson Xavier NX board is not able to handle the amount of computation necessary or if it is unable to obtain the desired frames per seconds, we plan on utilizing AWS Kinesis for real-time video processing and analysis. This is a backup plan in case we run into unforeseen problems with our board handling all of the compute power.

3.2 Software

The software approach of CarMa is split into two main cycles. When the user first picks up the device they will be asked to complete a calibration cycle. From then on, the user will be categorized as asleep or distracted vs attentive using machine learning algorithms discussed in a later paragraph. This is summarized and illustrated in the diagram in figure 5.

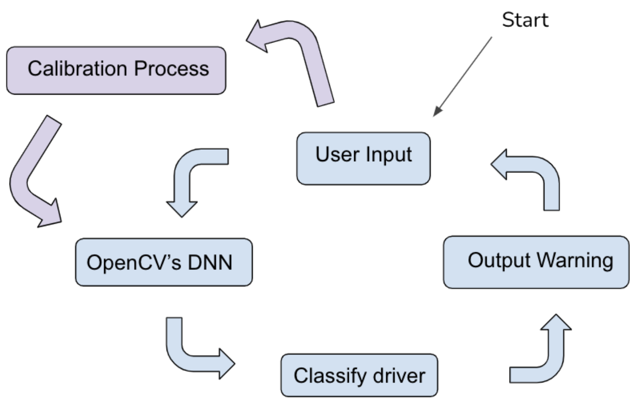


Figure 5: General Software cycle of CarMa

When a user first begins using CarMa, they will be asked to complete a calibration process, where the device can read their facial dimensions to more accurately categorize if the driver is sleepy or distracted. In particular, we are recording the distance of the lips when the mouth is closed along with eye measurements of the driver looking straight at the road. These will serve as our point of reference when checking if the driver is distracted or not. This is also to ensure the users have adequate background lighting which will improve the facial detection algorithms. Figure 6 illustrates a mock-up of the process.

Figure 6: Mock-up of calibration process

3.3 Computer Vision Algorithms

The computer vision aspect of CarMa allows us to detect whether or not the user appears distracted. The approach is summarized by the block diagram in Figure 7. We accomplish this task by first obtaining a facial detector followed my a landmark detector.

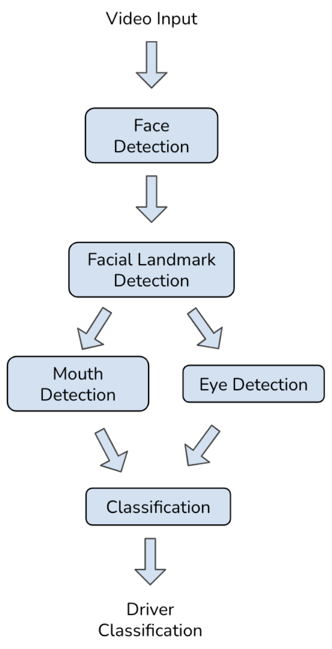


Figure 7: Block Diagram for the Computer Vision Algorithms

3.3.1 Face Detector

We are obtaining face detection by utilizing OpenCV and deep learning [5]. In particular, we are using OpenCV's "deep neural networks" (DNN) module which supports different learning frameworks such as TensorFlow and Caffe based face detectors. OpenCV's face detector is deep learning based and it utilizes the Single Shot-Multibox Detector framework with a ResNet base network.

The DNN module requires us to pass in .prototxt file which defines the model architecture along with the .caffemodel file which contains the weights for the actual layers. Both of these files allow us to read a network model stored in memory by utilizing cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"]). This returns an artificial neural network which allows us to pass the image through the network to obtain defections and predictions for the face. In short, we pass the image through the network (also known as forward propagation) to obtain the result (with no backpropagation). This gives us the face detection bounding box rectangles predictions.

3.3.2 Landmark Detector

In order to obtain our eye and mouth detection we obtain facial landmarking. In particular, we are utilizing a Convolutional Neural Network for facial landmarks detection, CNN-facial-landmarks [4], as a landmark detector that estimates the location of 68 points that map to a facial structures on the face. We can then load and build the facial landmarks model as Tensorflow model. We are then able to find the facial landmarks in an image from the faces by passing the landmark model along with the predicted face detection rectangles into the facial keypoint detector to find the facial landmarks in an image from the face.

3.3.3 Eye and Mouth Detection

In order to track the eyes we obtain the correct landmarks corresponding to each left and right eye. We then create a region of interest on a mask with the size of the driver's eyes and also find the extreme points of each eye. The mask is the same dimensions as the frame. Using the mask, we then segment out the eyes from the image. After we segment out the eyeballs from the rest of the eye we then find its center. We can then track the eyeball movements along with indicating if the center of the eyeball has changed and reporting if the driver is looking away from the road which we calculated during the calibration process. This eye tracking method will continue for each frame in a video sequence. A similar approach is done for mouth detection. However, this method requires finding the drivers mouth and recording the distance between the lips and comparing that distance during the calibration step when the mouth was closed. A open mouth means that the driver is yawning and potentially beginning to get drowsy.

To obtain more accurate results of the eye and mouth detection we follow more precise thresholding processing steps namely erosion, dilation, and median blurs.

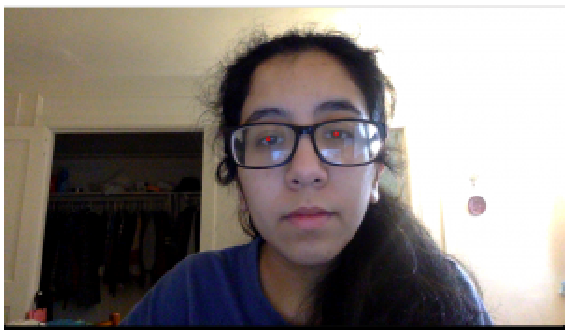


Figure 8: Eye Tracking

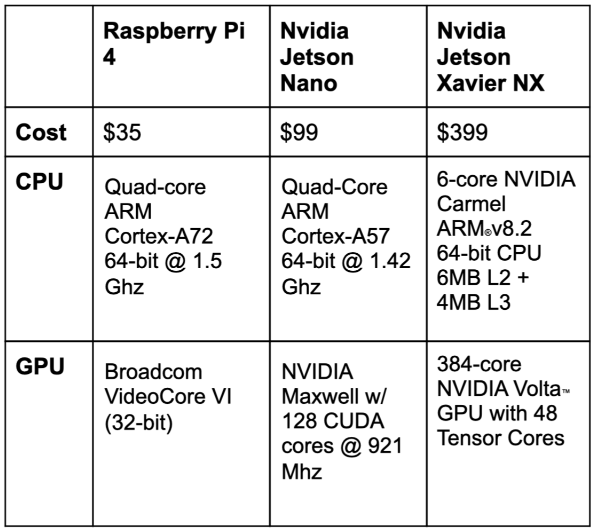
4 DESIGN TRADE STUDIES

A number of design tradeoffs were made throughout the design process of our system. The most important tradeoffs involved the embedded board we chose and the computer vision algorithms we picked. In this section, we will discuss the following trade-offs:

- Embedded Board
- Detection Models
- Scope: Lighting and Shadows
- Scope: Speeds
- Scope: Pose Detection
- Dataset Bias

4.1 Embedded Board

There were a number of boards that we looked at in our process of choosing an embedded board. Our requirements were that we wanted a board powerful enough to run our computer vision application at greater than 5 frames per second. Another requirement for our system was that it should be as compact of an overall system as possible. Thirdly, the system should support the necessary sensor inputs that we have. Lastly, the board should be less than $500 due to our $600 budget. From our research, we found 3 suitable boards that would meet the I/O, size, and budget requirements. A chart of the comparisons between the three is shown below.



	Raspberry Pi 4	**Nvidia Jetson Nano**	**Nvidia Jetson Xavier NX**
Cost	$35	$99	$399
CPU	Quad-core ARM Cortex-A72 64-bit @ 1.5 Ghz	Quad-Core ARM Cortex-A57 64-bit @ 1.42 Ghz	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6MB L2 + 4MB L3
GPU	Broadcom VideoCore VI (32-bit)	NVIDIA Maxwell w/ 128 CUDA cores @ 921 Mhz	384-core NVIDIA Volta™ GPU with 48 Tensor Cores

Figure 9: Embedded Board Comparison

We decided to go with the Nvidia Jetson Xavier NX board due to its onboard GPU and high compute. The Xavier NX was the most expensive out of the three boards so we had to budget our other parts accordingly. We would not be able to get expensive sensors and needed to borrow from the ECE parts inventory as much as possible.

4.2 Detection Models

Face detection is the most crucial aspect of the project. There are multiple pre-trained models available for face detection and the two that we focused on include OpenCV's DNN Module vs Dlib frontal face detector.

OpenCV's DNN Face Detector is a Caffe model based on the Single Shot-Multibox Detector (SSD) and uses ResNet-10 architecture as its backbone. The Dlib toolkit contains machine learning algorithms useful for computer vision. Dlib is used for face detection and facial landmark detection. The frontal face detector is based on histogram of oriented gradients (HOG) and linear SVM. The frontal face detector works by using features extracted by HOG which are then passed through an SVM.

We found that Dlib algorithms did not perform well for side faces. OpenCV's DNN module was able to detect side faces and quick head movement was not an issue. Dlib failed at large angles and quick movement. In addition, OpenCV's DNN module ran at about 12.95 frames per second while Dlib ran at about 5 frames per second, [1] so in terms of speed OpenCV's DNN module was the best choice.

Since OpenCV's DNN module proved to be a better facial detection module for our scope, we decided to use CNN-facial-landmarks [4], as a landmark detector that estimates the location of 68 points that map to a facial structures on the face. As opposed to the pre-trained facial landmark keypoint detector inside the Dlib library.

4.3 Scope: Lighting and Shadows

Another major tradeoff we discussed was the scope of our project specifically in relation to the environment CarMa would operate under. Based on the algorithms and preliminary tests, we immediately realized lighting is a crucial factor. When a driver is wearing glasses and there is reflection from the sun, neither of our algorithms performed well. Therefore, we limited our scope to good lighting conditions where the driver's face is clear and visible. Our groups also confirmed this would be the best approach with Professor Marios Savvides.

4.4 Scope: Speeds

Another scope based trade-off we considered was the speed at which the vehicle is operating at. Based on the research conducted, the NHTSA identified that most fatal accidents occurred at speeds over 55 miles per hour. For this reason, we wanted to ensure that our device would accurately monitor the driver at these speeds. We decided that CarMa will prioritize these speeds and guarantee a high accuracy even if this accuracy is not guaranteed at lower speeds due to the risk of fatal accidents.

4.5 Scope: Pose Detection

While coming up with the goals for this project, our group decided that the primary goals should be identifying and detecting distracted eyes and a sleepy driver. Distracted eyes would be identified if the driver's eyes are off the road for over 2 seconds and a sleepy driver would be identified by long blinks, more frequent blinks, eyes closed or yawning. Pose detection would need to build on top of the facial land marking and eye detection already used. In addition, pose detection without the previously described goals would not be effective for our project. So, pose detection is an attainable stretch goal.

4.6 Dataset Bias

As we research which data sets to train our model on, we realize there are two main options. We could build our own data set with short clips of various drivers blinking and yawning. On the other hand, our model could use a data set found online such as DrivFacce [6]. Such a data set would mean less time spent towards building the data set but may require more time to adjust the videos to exactly what CarMa requires, specifically blinking and yawning. Building our own data set would require more time filming people in cars but we could build the data set based on the specifications we require. One downside is that this would have bias towards the test subjects we use to build the data set.

After careful consideration, we have decided to construct our own driver data set specific for the models CarMa utilizes and our use case.

5 SYSTEM DESCRIPTION

The CarMa system can broken down into four main subprocesses:

- Sensor Input
- Calibration
- Face Detection
- Facial Landmark Detection
- Eye and Mouth Detection
- Pose Estimation

5.1 Sensor Input

The system will be taking input from two main sensors. Foremost, the camera module will be capturing images which will be sent to the computer vision application. Secondarily, the accelerometer will collect data on the driver's speed which will adjust the strictness of the driver monitoring.

The camera will be used as soon as the system starts the calibration process. Once the user enables driver monitoring, both the camera and the accelerometer will be sampling data at a fixed rate.

We decided to go with a small, low resolution camera as we want the system to be compact and have fast image processing. With a lower resolution, our image processing with occur quicker. For the accelerometer, we decided to borrow one from the ECE Inventory as our system does not specify a need for high accuracy for speed.

5.2 Calibration

On startup, our application will load the calibration sequence, which is a series of prompts that ask the user to capture their face at different angles. We will use the images captured through the calibration process to determine parameters to use for our computer vision application. The first step of the calibration process is to take a front facing image of the driver. There will be a prompt on the touch screen asking the driver to position their face looking forward at the road and to press a capture photo button when they are ready. There will then be a 5 second countdown before the image is captured. We will then ask the user to turn their head to the left and repeat the process. Finally, we will repeat the process with the user facing to the right.

5.3 Face Detection

Once the calibration process is complete and the driver monitoring begins we utilize OpenCV's Deep Neural Network (DNN) with Caffe model as the deep learning framework. OpenCV's deep learning face detection based on Single Multi-Shot Detector framework allows us to take one single shot to detect multiple objects within the image. The object localization and classifications are done in a single forward pass of the network. OpenCV's Deep Neural Network has a ResNet base network which is a residual neural network trained on ImagNet.

5.4 Facial Landmark Detection

Once the driver's face is detected, CarMa moves on to identifying facial landmarks.

It is difficult to capture the frontal section of a human face in real life situations, therefore we utilize an additional step, "face alignment" after face detection. It is necessary to detect the feature points in the face image, and mark some specific areas such pupils, corner of the eyes, mouth location, and more. This feature point detection is known as "Facial landmark localization".

In our algorithms of facial landmarking, we are utilizing CNN-facial-landmark [4] which is a landmarks detection based on convolution neural network, this model is build with Tensorflow. The landmark detector returns 68 landmarks each related to a specific point on the face.

5.5 Eye/Mouth Detection

After the feature points are detected, we are able to mark the specific position of the face in the image. Utilizing masking, we are able to obtain only the parts of the face that we care about. In order to detect if the driver is distracted we can check if their eyes have been focused elsewhere for 2 or more seconds. Similarly to identify if the driver is sleepy, we will detect if the driver's blinks are longer or more frequent.

We will follow a similar process for mouth detection. Using masking, we will single out the area of the face necessary. In order to identify if the driver is sleepy, we will check if the driver is yawns, repeatedly.

5.6 Pose Estimation

Pose estimation is a stretch goal that will build off of the facial detection, facial land marking and eye detection already in place. Pose estimation will use the angle of the face and its features to determine the angle at which the driver is turned. One method we will try is to use the distance between the facial landmarks we produce to determine an angle. For instance if my face is turn the the left, the distance between the landmark of my left check and nose will be closer and the distance from my nose and my right check will be farther.

6 PROJECT MANAGEMENT

6.1 Schedule

The full Gantt chart of the schedule is located in Appendix A. The schedule contains each part of our project along with how we decided to break each task down. The schedule is divided by each member with each of their tasks listed. Adriana's tasks are in blue, Evann's are in pink, and Jananni's are in green.

6.2 Team Member Responsibilities

In terms of the division of labor, we have split up the responsibilities for each person as primary and secondary objectives. For the primary objectives, we focused on facial and eye detection. For the primary objectives, Adriana will be working on the facial detection. Evann will be working on building the hardware aspect of the device. Jananni will be focused on the eye detection.

For the secondary objectives, we focused on optimization and stretch goals. For the secondary objectives, Adriana will be working on algorithm optimization to make sure we are within our project requirements. Evann will be building off the facial landmarking and eye detection for the pose detection. Finally, Jananni will be working on the calibration and thresholding.

6.3 Budget

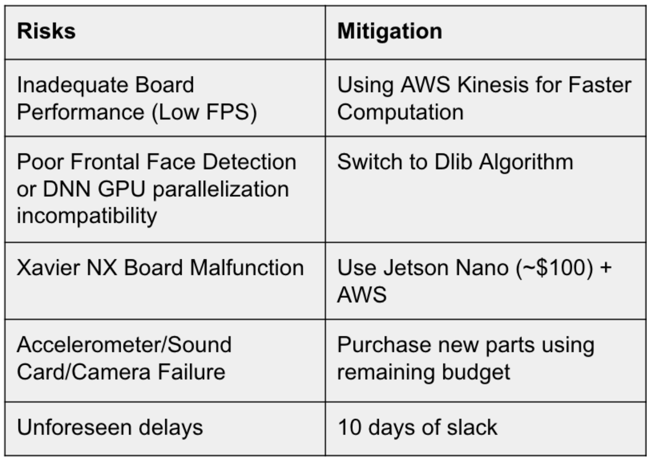
Table 1: Bill of Parts

Component	Cost	Notes
Jetson Xavier NX	$400	Arrived
Developer Kit	$83	Arrived
Accelerometer	-	Borrowed
Accelerometer	$9	Backup
Sound Card	-	Borrowed
Sound Card	$10	Backup
AWS Credit	-	ECE Free Credit
WiFi Card	$25	In Progress
Car Power Adapter	$13	In Progress
Total	$540	

CarMa requires multiple parts in order to successfully work. The majority of the budget is spent on the Nvidia Jetson Xavier NX board. The project also requires the use of a Developer Kit as that comes with a touch screen for the board, along with a camera and other additional gadgets. We have borrowed an accelerometer along with a sound card from the Capstone course but also accounted for spare parts in our budget in case the parts break or malfunction. Additionally, we have accounted for a backup plan of utilizing AWS Kinesis for real time data streaming in the case that the board cannot handle all of the compute power. Thus, this would require a WiFi card in order to connect to the internet to perform computation through AWS. Lastly, when taking the board out for testing, a car power adapter would be required in order to charge the board.

6.4 Risk Management

In order to ensure that everything will run smoothly, we have put together a list of potential risks that could occur. Below is a full table of the risks and our corresponding management plans.



Risks	Mitigation
Inadequate Board Performance (Low FPS)	Using AWS Kinesis for Faster Computation
Poor Frontal Face Detection or DNN GPU parallelization incompatibility	Switch to Dlib Algorithm
Xavier NX Board Malfunction	Use Jetson Nano (~$100) + AWS
Accelerometer/Sound Card/Camera Failure	Purchase new parts using remaining budget
Unforeseen delays	10 days of slack

Figure 10: List of potential risks and corresponding mitigation plans

One of the biggest risks that is crucial for our project is that the Nvidia Board is inadequate for our use case. The mitigation plan for the computation risk is using AWS Kinesis for video streaming analysis. This will take the weight off the board and will speed up our computation time and frames per second.

In a related scenario, if the Nvidia Jetson Xavier NX is pushed to its capacity and malfunctions, the back-up plan is to utilize the Nvidia Jetson Nanos that the ECE department has in addition to AWS Kinesis which should handle most of the heavy computation.

In case our Frontal Face detection algorithm is not accurate enough to meet our requirements or is incompatible with our hardware, the mitigation plan is to switch to the slower but more accurate Dlib Algorithm which works very well for the frontal view but not well for side view.

Our group has also accounted for the case when specific components fail such as the accelerometer, sound card or camera. Currently, we are borrowing the accelerometer and sound card from the ECE department and our Nvidia Development Kit comes with a camera. If any of these components malfunction, we have a portion of our budget saved so that we can buy additional components.

In addition to the additional money saved for spare components, we have also planned our schedule such that there will be additional slack days for unforeseen delays and issues. Most likely this will be due to the integration and testing phases.

7 RELATED WORK

One commercial product that is similar to our project is Valeo. This product has a camera embedded behind the steering wheel leading to a very compact system. The product has three main functionalities:

1. Identification of the driver
2. Monitor driver fatigue and trigger alert
3. Monitor driver attentiveness by tracking eyes

Being a commercial product, the number of features that the system has implemented is limited. Also, as a private institution, Valeo hasn't released much of its confidential research.

A research project that we found based some of our project off of is the paper on *Drowsy driver detection system using eye blink patterns* [3]. The researchers used a standard webcam and was able to achieve a frame rate of 110 fps and accurately detect blinks. The paper compares the blink duration between drowsy state and alert state to determine when a driver begins to get sleepy. We do not expect to get a high enough frame rate to be able to detect most blinks but if we are able to optimize our computer vision application to have a frame rate greater than 15 fps, we will explore detecting long blinks to determine drowsiness.

7.1 Future Work

One feature that would build off the current project is a phone detection feature. If CarMa could identify a cellphone in frame then the device could alert the driver about a possible distraction.

Another idea for CarMa is lane change detection. The device would remind the driver to turn on their signals if the blinker sound is not detected and would remind the driver to check their blind spots if the driver does not do a shoulder check.

A last future idea was to identify when the car was parked at an angle and remind the driver to use their handbrake.

References

[1] Vardan Agarwal. "Face Detection Models: Which to Use and Why?" In: *https://towardsdatascience.com/face-detection-models-which-to-use-and-why-d263e82c302c* (July 2020).

[2] CDC. "Distracted Driving". In: *CDC* (Dec. 2020).

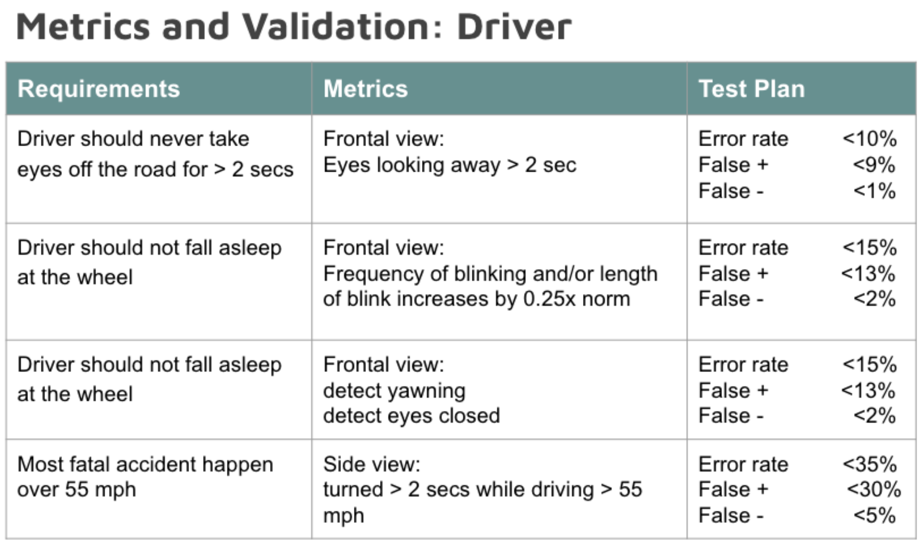
[3] Taner Danisman. "Drowsy driver detection system using eye blink patterns". In: *IEEE* (Oct. 2010).

[4] Yin Guobing. “CNN Facial Landmarking”. In: *https://github.com/yinguobing/cnn-facial-landmark* (Dec. 2017).

[5] Adrian Rosebrock. “Face detection with OpenCV and deep learning”. In: *pyimagesearch* (Feb. 2018).

[6] ElektraAutonomous Vehicle. “DrivFace Data Set”. In: *http://adas.cvc.uab.es/elektra/enigma-portfolio/cvc11-drivface-dataset/* (Apr. 2016).

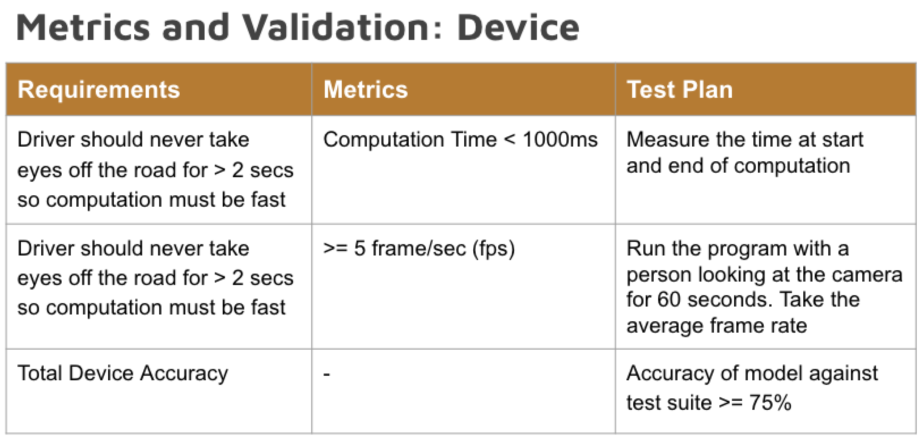
Appendix A



Metrics and Validation: Driver

Requirements	Metrics	Test Plan
Driver should never take eyes off the road for > 2 secs	Frontal view: Eyes looking away > 2 sec	Error rate <10% False + <9% False - <1%
Driver should not fall asleep at the wheel	Frontal view: Frequency of blinking and/or length of blink increases by 0.25x norm	Error rate <15% False + <13% False - <2%
Driver should not fall asleep at the wheel	Frontal view: detect yawning detect eyes closed	Error rate <15% False + <13% False - <2%
Most fatal accident happen over 55 mph	Side view: turned > 2 secs while driving > 55 mph	Error rate <35% False + <30% False - <5%

Figure 11: A table of the Driver Metrics



Metrics and Validation: Device

Requirements	Metrics	Test Plan
Driver should never take eyes off the road for > 2 secs so computation must be fast	Computation Time < 1000ms	Measure the time at start and end of computation
Driver should never take eyes off the road for > 2 secs so computation must be fast	>= 5 frame/sec (fps)	Run the program with a person looking at the camera for 60 seconds. Take the average frame rate
Total Device Accuracy	-	Accuracy of model against test suite >= 75%

Figure 12: A table of the Device Metrics

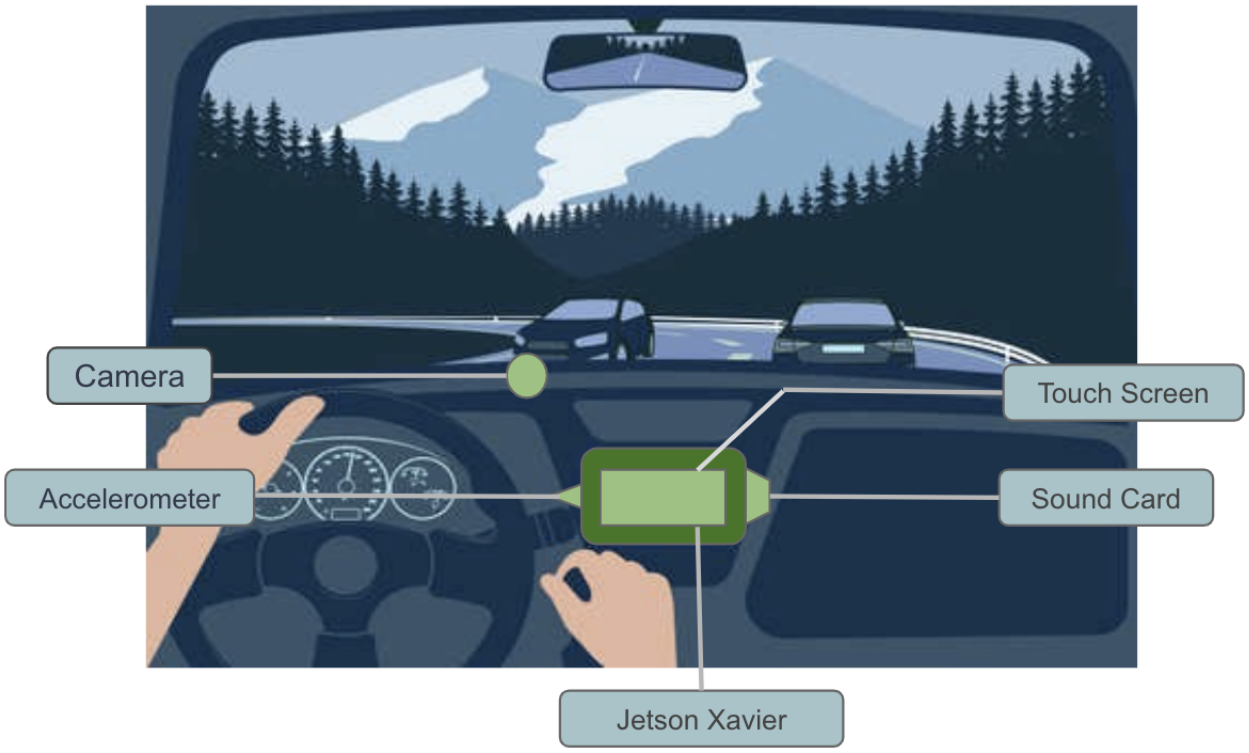


Figure 13: Mock-up of our Device

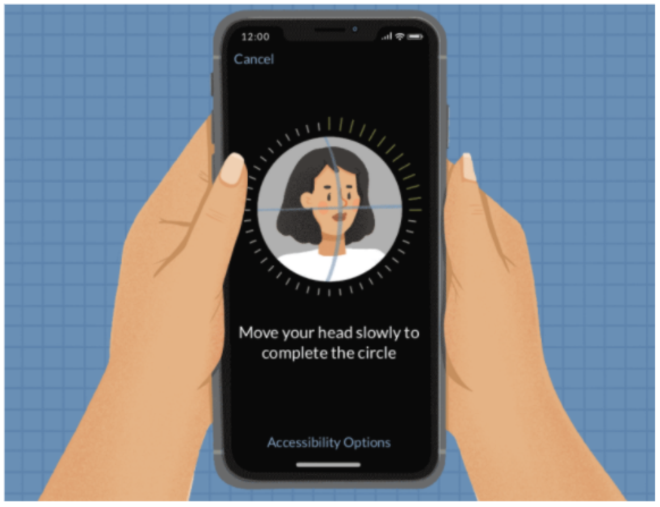


Figure 14: Mock-up of calibration process

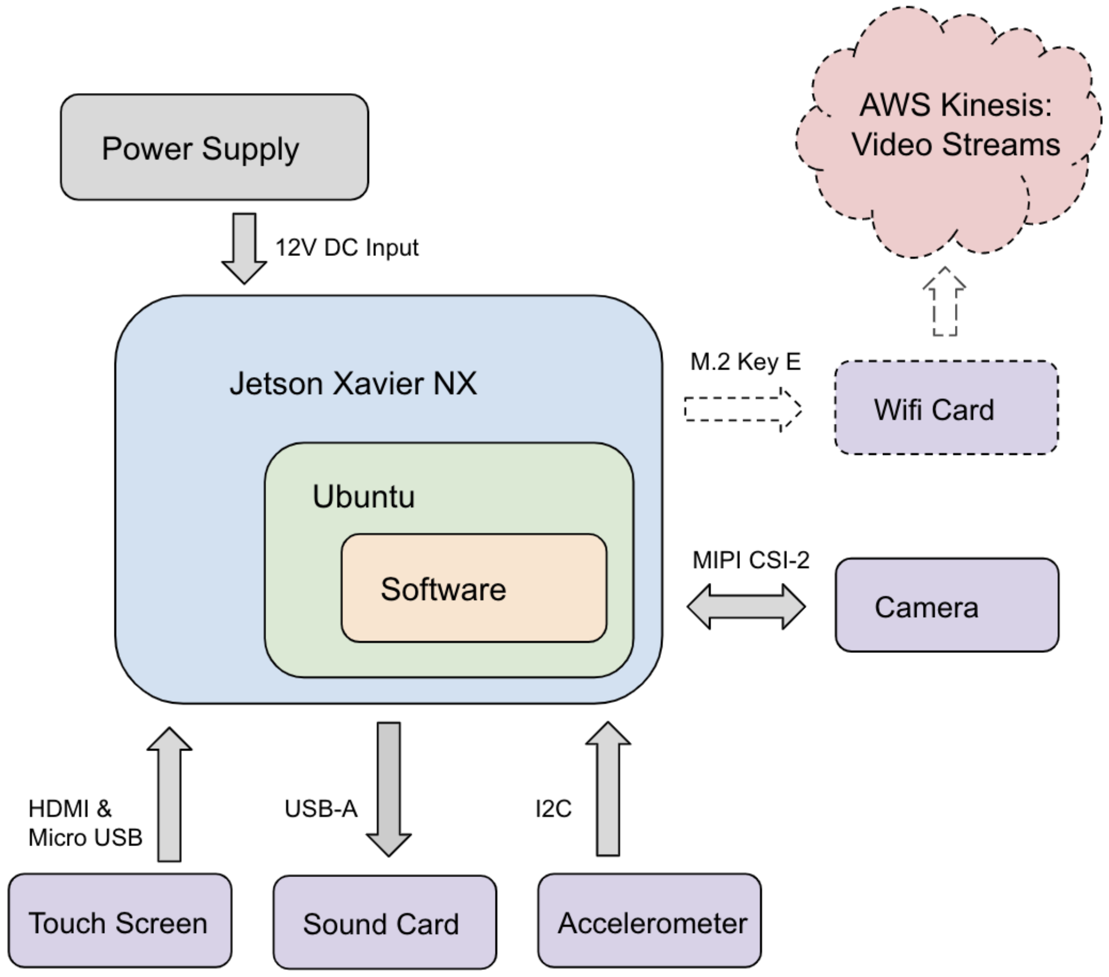


Figure 15: Hardware architecture of CarMa

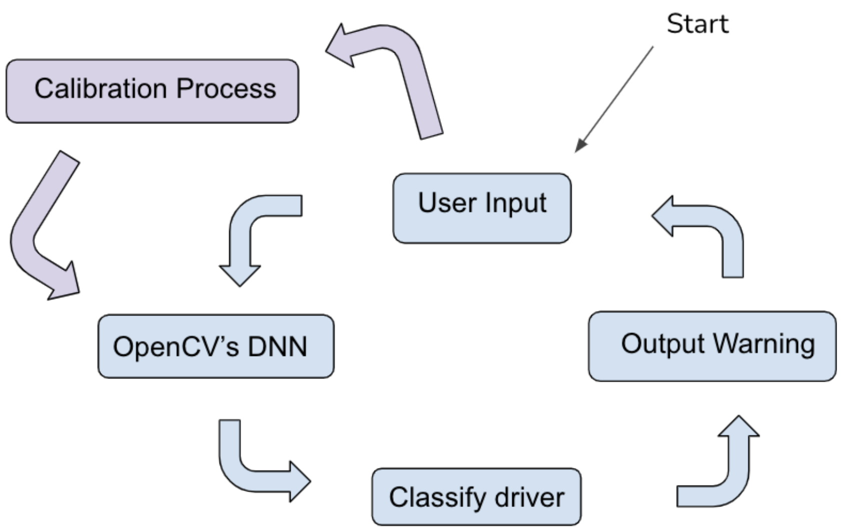


Figure 16: General Software cycle of CarMa

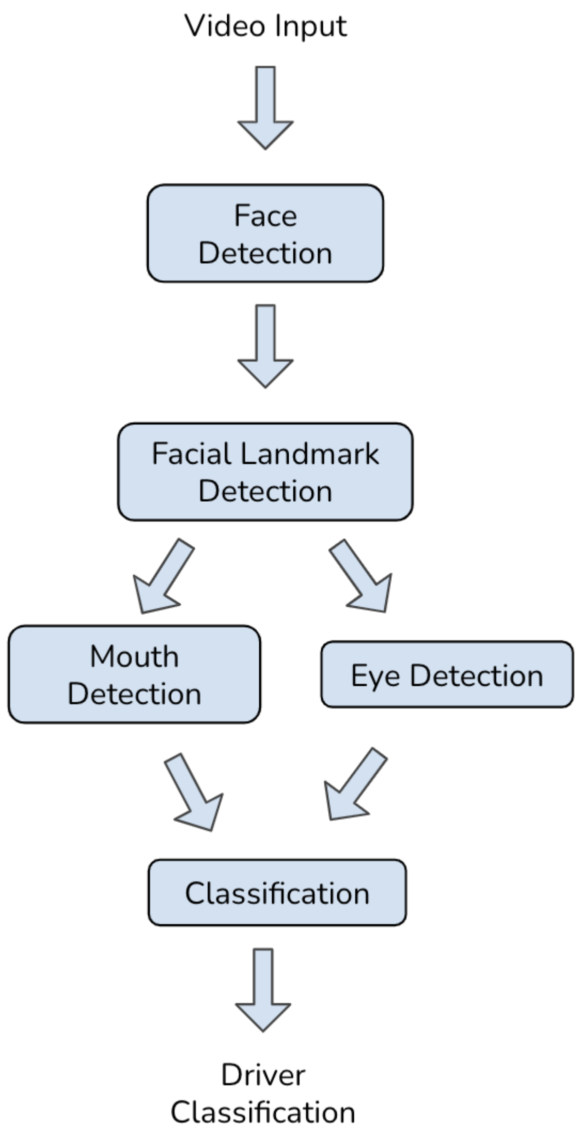
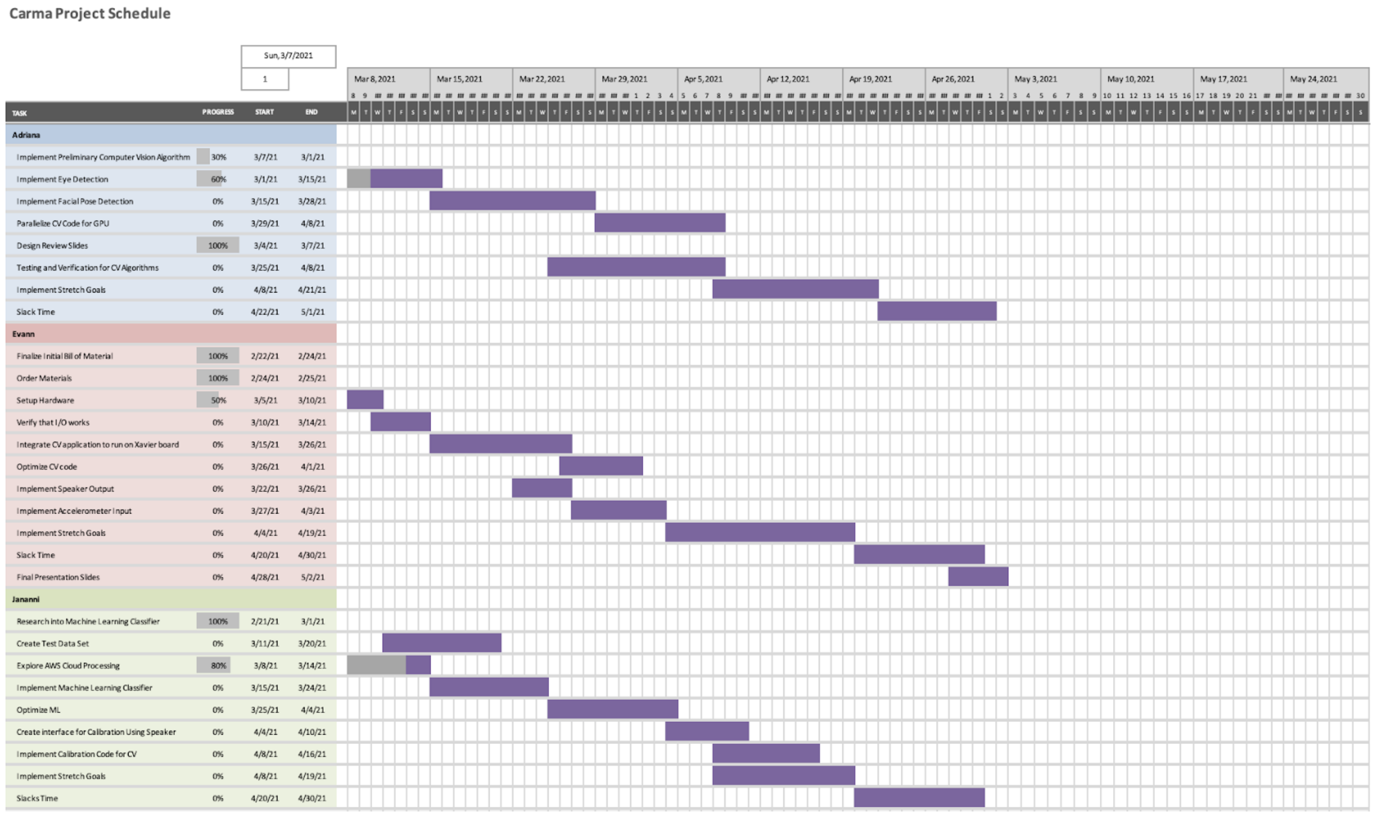


Figure 17: Block Diagram for the Computer Vision Algorithms



TASK	PROGRESS	START	END
Adriana			
Implement Preliminary Computer Vision Algorithm	30%	3/7/21	3/1/21
Implement Eye Detection	60%	3/1/21	3/15/21
Implement Facial Pose Detection	0%	3/15/21	3/28/21
Parallelize CV Code for GPU	0%	3/29/21	4/8/21
Design Review Slides	100%	3/4/21	3/7/21
Testing and Verification for CV Algorithms	0%	3/25/21	4/8/21
Implement Stretch Goals	0%	4/8/21	4/21/21
Slack Time	0%	4/22/21	5/1/21
Evann			
Finalize Initial Bill of Material	100%	2/22/21	2/24/21
Order Materials	100%	2/24/21	2/25/21
Setup Hardware	50%	3/5/21	3/10/21
Verify that I/O works	0%	3/10/21	3/14/21
Integrate CV application to run on Xavier board	0%	3/15/21	3/26/21
Optimize CV code	0%	3/26/21	4/1/21
Implement Speaker Output	0%	3/22/21	3/26/21
Implement Accelerometer Input	0%	3/27/21	4/3/21
Implement Stretch Goals	0%	4/4/21	4/19/21
Slack Time	0%	4/20/21	4/30/21
Final Presentation Slides	0%	4/28/21	5/2/21
Jananni			
Research into Machine Learning Classifier	100%	2/21/21	3/1/21
Create Test Data Set	0%	3/11/21	3/20/21
Explore AWS Cloud Processing	80%	3/8/21	3/14/21
Implement Machine Learning Classifier	0%	3/15/21	3/24/21
Optimize ML	0%	3/25/21	4/4/21
Create interface for Calibration Using Speaker	0%	4/4/21	4/10/21
Implement Calibration Code for CV	0%	4/8/21	4/16/21
Implement Stretch Goals	0%	4/8/21	4/19/21
Slacks Time	0%	4/20/21	4/30/21

Figure 18: Schedule Gantt Chart