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 use computer vision to detect if the driver is falling asleep. Our aim is to prevent drivers from falling asleep at the wheel.**
CarMa primarily focuses 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 day light on the user's face.

***Index Terms*—Driving assistant, Computer Vision, Facial Detection, Facial Landmarks, Eye tracker, Mouth Tracker, Machine Learning, 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 adults learning how to drive for the first time. We believe it is critical that new drivers learn safe driving habits as they begin driving. 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 an explored field, 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 user friendly.

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 efficient 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 and as close to real time as possible.

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 which is their requirement for focus driving. 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: detect yawning detect eyes closed	Error rate <15% False + <13% False - <2%
Most fatal accident happen over 55 mph - stretch goal	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
- Eyes closed for over 2 seconds
- Yawning - mouth open for over 2 seconds

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 fps	Measure frame rate with person looking at the camera for 60 seconds
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 loosely follows 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

There are multiple hardware components that CarMa needs. 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 Xavier 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 Jetson 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 components of this project will be connected.

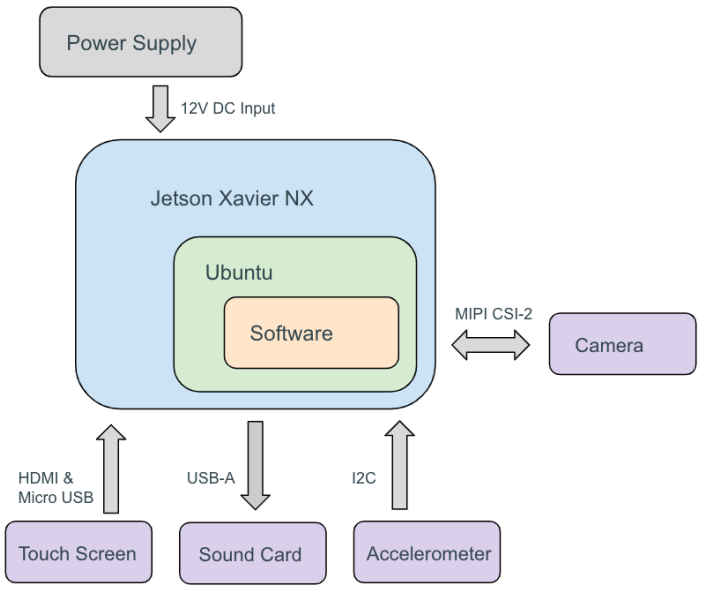


Figure 4: Hardware architecture of CarMa

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 process. From then on, the user will go into the main cycle (depicted in blue in Figure 5) where they will be categorized as drowsy or distracted vs attentive using machine learning algorithms discussed in the later sections. This entire process is summarized and illustrated in Figure 5.

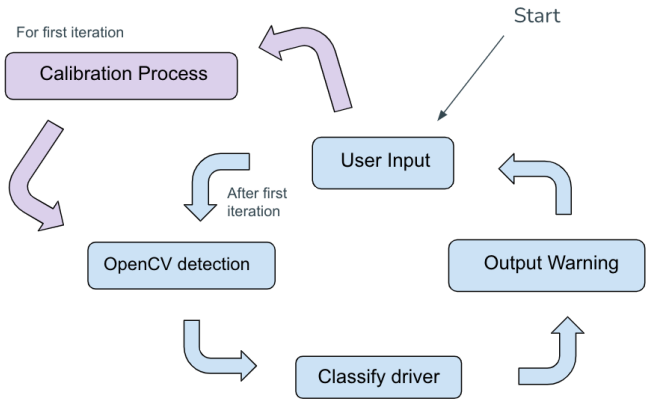


Figure 5: General Software cycle of CarMa

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. We ask the user to face the camera straight ahead with their eyes focused and their mouth open wide. In particular, we are recording the vertical distance of the lips when the mouth is wide open. We also record the eye aspect ratio when the driver is 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 that we have followed.

Figure 6: Mock-up of calibration process

3.2.1 User Flow

Users will be interacting with CarMa through our touch screen. The touch screen consists of the following pages shown in Figure 7. At the start, the user sees the starting screen page, to continue they press the "start" button. Then they are directed to the calibration page. Once they click on "start" button on the calibration page then the calibration process begins. This process requires the user to place their head inside the circle that is shown to them. After their head placement is correct then it automatically takes the user to the page where the user is ready to start driving. To begin the CarMa alert program, they press "start" and the program is now classifying when the user appears to be distracted or not! The driver can easily end the program by pressing the "stop" button and they will be redirected back to the start screen.

Figure 7: CarMa User Flow

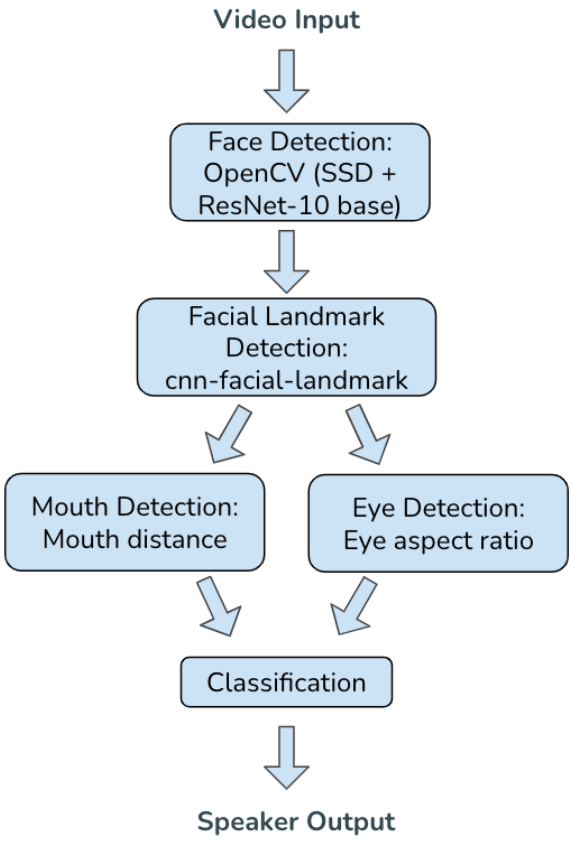


Figure 8: Block Diagram for Computer Vision Algorithms

3.3 Computer Vision Algorithms

The computer vision aspect of CarMa allows us to detect whether or not the user appears to be distracted. The approach is summarized by the block diagram in Figure 8. We accomplish this task by first obtaining a facial detector followed my a facial landmark detector in order to detect mouth and eyes. This then allows us to form a classification to determine if the user is distracted.

3.3.1 Face Detector

The first part of our algorithm is to detect a face in an image. Each frame of the video captured is used for face detection in images using OpenCV and deep learning. OpenCV's deep learning "deep neural networks" (DNN) module [5] supports different learning frameworks and OpenCV's deep learning face detector is based on the Single Shot-Multibox Detector framework with a ResNet-10 base network as its backbone.

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 and it returns an artificial neural network which allows us to pass the image through the network to obtain defections and predictions for the face. We utilize this pre-trained model for the face detection in each frame. In short, we pass the image through the network (also known as forward propagation) to obtain the result (with no back-propagation). This gives us the bounding box rectangle for the face it detects. In the case where there are multiple faces detected in an image, then we use the face with the largest confidence.

3.3.2 Landmark Detector

Figure 9: Landmarks Example

In order to obtain our eye and mouth detection we obtain facial landmarks which are used to localize and represent regions of the face. Now that we have our bounding box prediction of where the face is located on the image, we then pass that information to our facial landmark detector. In particular, we are utilizing a Convolutional Neural Network based facial landmarks detector, 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 able to find the facial landmarks in an image containing 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. This is a crucial step in eye and mouth detection.

3.3.3 Eye Detection

Now that we have the facial landmark information on the given image, we are ready to begin our 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 then 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. Most importantly, we can track if the user has their eyes closed by comparing their current eye aspect ratio (EAR) to the eye aspect ratio they obtained during the calibration step. The eye aspect ratio is a constant value when the eye is open, but rapidly falls close to 0 when the eye is close. This eye tracking method will continue for each frame in a video sequence.

3.3.4 Mouth Detection

A similar approach is done for mouth detection. This method requires finding the drivers mouth and recording the distance between the lips and comparing the distance during the calibration step when the mouth was wide open. A open mouth means that the driver is yawning and potentially beginning to get drowsy. For mouth detection, we use the vertical mouth distance to determine if the mouth is open or closed. 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 10: Eye and Mouth Detection

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 in Figure 11.

	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 11: 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 pre-trained 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 support vector machine (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 group 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. Safety is the main goal of CarMa therefore make sure to account for the speed that the user is driving at in our program. We also wanted to avoid frequent alerts when the user is not moving the vehicle such as when the user is parked.

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 that we have set.

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. The data that we utilized is composed of videos from our friends and family.

5 SYSTEM DESCRIPTION

The entire CarMa system can broken down into two main sub-systems: the hardware components and the software components.

For Hardware we have:

- Sensor Input
- System Components

For Software we have:

- Calibration
- Face Detection
- Facial Landmark Detection
- Eye and Mouth Detection
- Pose Estimation
- Classification

5.1 Hardware

5.1.1 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.

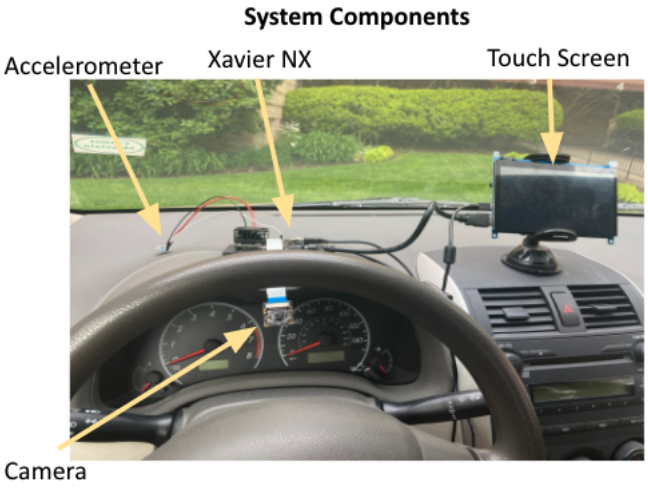


Figure 12: CarMa Components

5.1.2 System Components

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 will occur quicker. After some research, we settled on the IMX219-77 camera module as it could both capture low resolution images, 1280x720, and took up a very small amount of real estate, 25mm x 25mm. The IMX219-77 also interfaces easily with the Xavier NX's MIPI-CSI port. In terms of video capturing, we use a gstreamer pipeline to start video capture through our camera, and send that to our python program. This pipeline allows us to modify elements of the video capture such as frame-rate, orientation, and resolution.

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. The specific accelerometer module we received was the MPU-6050. This was both a fast and small module that fit the needs of our project. We connected the accelerometer to the board via I2C through the GPIO pins. Then we simply read the memory address where the sensor is mapped to every loop of our main program to update the speed of the car using the x-axis acceleration and the time between each sample. Only once

we detect that we are past 55 mph do we enable the driver monitoring.

Figure 13: Complete CarMa System

5.2 Software

Our software algorithms can be broken down into a few main parts. We first obtain the video input which is captured by the camera which is used for the following sections.

5.2.1 Calibration

Before the user is ready to receive warning sounds from CarMa, the user needs to go through the calibration process. 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. This process requires the user to place their head inside the shown circle with their mouth open and eyes open. After their head placement is correct then CarMa automatically takes their picture and the user is ready to start driving. We use the image captured through the calibration process to determine the parameters used for our computer vision application. In particular, we store the vertical height of the open mouth along with the eye aspect ratio of the eyes. These values allow us to better classify when the user might have their eyes closed or when the user is yawning.

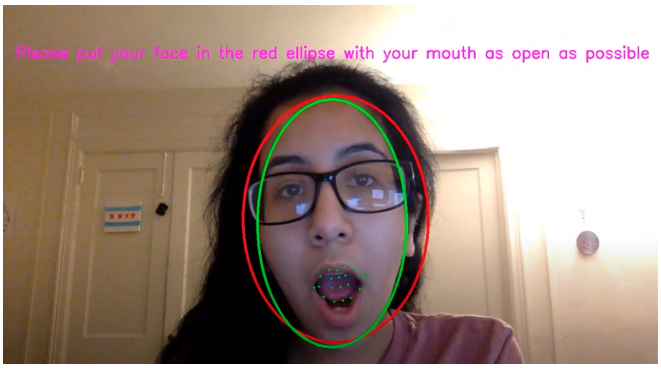


Figure 14: Calibration Process

5.2.2 Face Detection

Once the calibration process is completed, the user is ready to begin CarMa! The video input captured by the camera starts when the user clicks on the "start" button in the touch screen. Each frame of the video captured is used for face detection in images using OpenCV and deep learning. We utilize OpenCV's Deep Neural Network (DNN) with Caffe model as the deep learning framework. OpenCV's deep learning face detector is based on the Single Shot-Multibox Detector (SSD) framework with a ResNet-10 base network as its backbone. It 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. The ResNet-10 base network is a residual neural network trained on ImagNet.

5.2.3 Facial Landmark Detection

Once the driver's face is detected, CarMa moves on to identifying facial landmarks. The facial landmarks are used to localize and represent regions of the face such as eyes, nose, chin, and more.

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 detector based on convolution neural network. The facial landmarks detector gives 68 landmarks each related to a specific point on the face and it is a Tensorflow CNN trained on multiple data-sets. We utilized this pre-trained model for our project.

5.2.4 Eye and 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.

5.2.5 Distracted Eyes and Pose Estimation

Distracted eyes is when the user is not looking straight at the road. This is when their eyes are to the side either left, right, up, or down. 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.

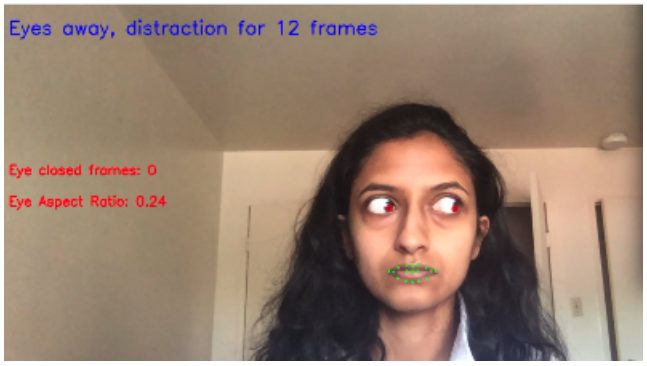


Figure 15: Distracted Eyes

5.2.6 Classification

We use the data obtained from mouth and eye detection along with pose estimated into a classifier to tells us if the user appears distracted. For eye detection, we classify the user as having closed eyes if their eye aspect ratio falls below 0.75 their eye aspect ratio calculated at the calibration step. In order to classify the user as having their eyes closed then they have to fall below this threshold for 12 consecutive frames which equals around 2 seconds. The frame constraint was put in place in order to avoid declassifying blinks as the user falling asleep.

For mouth detection, we classify the user as having an open mouth if their mouth height is greater than 0.90 of their open mouth height calculated at the calibration step. In order to classify the user as yawning then they have to fall within this threshold for 12 consecutive frames. The frame constraint was put in place in order to avoid declassifying talking as the user yawning asleep.

For distracted eye detection, we check to see if their eyes are looking to the side. If the user is looking to the side for 12 consecutive frames then we classify them as not looking at the road and therefore they are distracted.

If the user appears to be distracted, an alert will sound signaling to the user that they should adjust their behavior.

6 TEST & VALIDATION

Once we completed our system, we started working on a testing procedure to evaluate if our system meets our project requirements. To measure the accuracy of our system, we created a test suite of videos of various people blinking, talking, yawning, closing their eyes for over 2 seconds and looking in various directions for over two seconds. With these videos we kept track of how well our system did by counting the number of error events. We found that we met most of our original system requirements but there could definitely be improvements in relation to the eye detection. Our system was especially good at the face and mouth detection and we managed to achieve all our hardware requirements. We also kept track of our false positives and false negatives to ensure we are prioritizing false positives over false negatives because we would rather warn drivers more rather than less.

Metrics, Validation & Testing

Metrics	Test Plan	Test Results	Status	Prioritize False Positives
Frontal view: detect yawning	Error rate <20%	= 1%		
Frontal view: detect eyes closed	Error rate <30%	= 20.6%		
Frontal view: Eyes looking away	Error rate <40%	= 40%		
Side view: turned > 2 secs	Error rate <35%	Stretch goal		
Latency < 1000ms	Measure the time between action and output	183.3 ms		N/A
>= 5 fps	Measure frame rate with person looking at camera for 60 secs	5.70 fps		N/A
Error Rate of Algorithm	Accuracy of model against test suite >= 75%	85.9%		N/A

Completed | Very Close | Needs Work | Not Tested

Figure 16: List of requirements and their statuses

6.1 Results for Design Specification: Software

For the software aspect of this project, we did a lot of research for various aspects of the project including the Calibration process and more. But we needed to test and validate that the face, mouth and eye detection were good enough for our requirements.

6.1.1 Face Detection

OpenCV's DNN module performed very well in our testing. We performed 300 tests out of which the algorithm correctly detected the face 295 times. The accuracy was 98.3% which is well above what we needed for our project.

In our testing we did ensure that there was at least 1 face in the frame. We also defined an accurate detection as detecting any of the faces in the frame not necessarily the person running the program. But when in use, the video is angled so that only the driver's full face will be visible.

6.1.2 Mouth Detection

For the mouth detection, our algorithm still did quite well with only 1.64% error and 98.4% accuracy. Out of 61 total events, the program had only 1 error which far surpasses our initial requirement.

We tested 2 actions in these events. The first was yawns and the second was talking. If the program identified talking as a yawn we classified it as a false positive and if it did not classify yawning then it was a false negative. The one error for mouth detect was a false positive which is what we want to prioritize.

6.1.3 Eye Detection

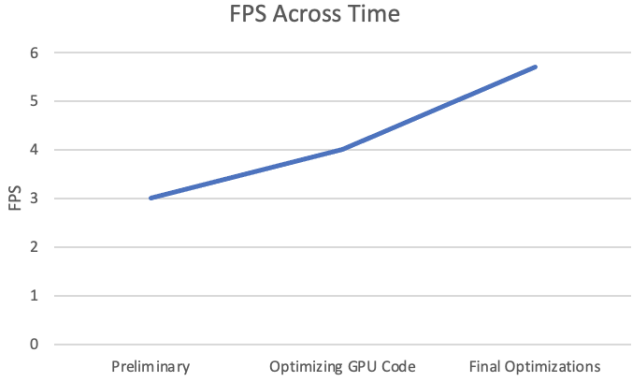
For the eye detection we had two actions we wanted to test. First we needed to test if the driver's eyes were closed for over 2 seconds. This was our first priority along with the mouth detection. Next we also tested if the driver's eyes were looking away from the road for over 2 seconds.

For the eyes closed for over 2 seconds, we had a total of 68 events with 14 resulting in an error so a 20.6% error rate. This means our eyes closed detection was 79.4% accurate. This meets our original requirements. Here we also satisfied our priority of false positives because we had 13.3% of our errors be false positives while only 7.4% were false negatives.

For the eyes looking away for over 2 seconds, we did less testing as we finished this later on in our project timeline. We had a total of 10 events, 4 of which were error events. So our eyes looking away error rate is 40% and accuracy is 60%. This was very close to what we were aiming to get and we are overall satisfied with the results for eye detection.

6.2 Results for Design Specification: Hardware

For testing our hardware, the main metric that we wanted to test was the frames per second that we attained from running our program on the Xavier NX. We measured this number continuously through the development of the project to establish our program's performance with every new update. The frames per second is collected by including a timer in our main while loop that will update every cycle.



7 PROJECT MANAGEMENT

7.1 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.

7.2 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.

7.3 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
Car Power Adapter	$13	Arrived
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.

7.4 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 17: 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.

8 ETHICAL ISSUES

In terms of ethical issues, one main concern was making sure the user was fully aware of what the system was doing. Since we utilize a camera, it is important to notify the user when we are recording. Similarly, we specify that we are not storing any of their footage. We also made it clear to the user that they are allowed to turn the system on and off in order to stop the application once they are done driving.

The clarity in our system was done through the UI and user flow of the touch screen. This involved creating on click functionality when the buttons on each page are clicked by the user. At the start, the user sees the starting screen page, to continue they press “start”. Then they are directed to the “calibration” page. Once they click on “start calibration” button then the calibration process starts. This requires the user to place their head inside the circle. After their head placement is correct then it automatically takes the user to the page when the user is ready to start driving. To begin the CarMa alert program, the user presses the “start” button and the program is now classifying when the user appears to be distracted or not. This design flow was based on the ethics conversation with students and professors where it was mentioned that it should be clear when the program is starting and recording so the user can have a clear understanding of the program.

More feedback we received included the placement of the touchscreen. Since our aim is to avoid distracted driving, we carefully placed the touch screen in a position where it would not be distracting the driver as that would defeat the whole purpose of our product. It is in the same location a smart phone or a GPS would be located in the car. We also made sure that the UI was verbose enough to be clear but simple enough to be non distracting. When the user is driving the screen we show in the touch screen remains static until the user clicks the ”stop” button.

A possible edge case with CarMa is the failure to properly classify people with smaller eyes. This is because our eye tracking algorithm checks to see if the user’s eyes are closed. If the user has small eyes then our algorithm might not be able to accurately detect when they are closing their eyes. Therefore people with smaller eyes would be affected the most by this edge case. In order to mitigate any potential foreseen adverse impact we could improve our machine learning model to have more robust classifications by testing it with more pictures of smaller eyes.

9 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.

Another commercial product that we looked into was Rosco's Dual Vision XC4. This device records the driver which the driver can look back on to see any unsafe behavior. It also alerts drivers if they are going over the speed limit and or if it detects a G-force that is over a specified threshold. This product does not do any real time drowsy or distracted driver monitoring, which is our main focus, however it has many of the same hardware sensors that we do and has a similar application.

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 did not get a high enough frame rate to be able to detect most blinks but if we were able to optimize our computer vision application to have a frame rate greater than 15 fps, we could have explored detecting long blinks to determine drowsiness.

We also found some research papers which performs facial pose detection. *Face Detection, Pose Estimation, and Landmark Localization in the Wild*[7] describes a pose detection method using facial landmarks and SVM to train a model to detect pose. This method was found to be extremely accurate, scoring a 99% accuracy for MultiPIE data-set which is a common benchmark for this type of analysis.

10 SUMMARY

To reiterate, CarMa is a driving assistant meant to remind the driver to focus on the road. Our project takes in real-time video input and classifies the driver as sleepy/distracted using their mouth and eyes. We check for if the driver is yawning, eyes are closed for over 2 seconds or eyes are off the road for over 2 seconds.

Our system was able to meet almost all of our design requirements. We wanted CarMa to be able to accurately detect the driver's face and if they were sleepy. CarMa's face detection and mouth detection performed very well at 98.3% and 98.4% accuracy respectively. We knew eyes would be harder to detection and classify so we set our requirements lower and we met our eyes closed requirements and just barely missed our eyes looking away requirement.

If given more time, we would have worked to improve our eyes detection (eyes closed and eyes away) to be more accurate. One potential idea would be working on the thresholding to improve the contrast between the eyeball to improve masking.

Overall, against our test suite CarMa had an 85.9% accuracy across all our tests.

In terms of hardware, CarMa met our latency and fps requirements by far at 183 ms and 5.7 fps respectively.

One of the biggest roadblocks we faced while trying to increase our fps was utilizing the Jetson Xavier's GPU. Our team worked on this for a couple weeks and tried various things but were not able to use the GPU. Because we had already spent a lot of time and we were passed our required fps we decided to work on other things. But this would have been a big improvement to our computation speed.

Another improvement that could have helped for the eye classification is training the model we used rather than using the pre-trained model. This would mean every time the driver started our device and went through the Calibration process, we would use their image to train our model. In addition, we could have used videos specific to driving to train our model to make it more robust.

10.1 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. We currently detect if the users eyes are not looking at the road which can translate to if the user is possibly looking down at their phone however, we do not identify if the phone itself is in the frame.

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. Which is something that is easily forgettable for drivers who are not used to driving in hilly areas.

10.2 Lessons Learned

An important lesson that we learned working on this project is that you should design your system to be as robust as possible. This includes robustness for different users, different lighting conditions, different angles, and many other factors. Only when your system is robust to all these factors can you have a usable product.

Another lesson we learned was that you should start collecting data for testing as soon as you have an idea of what exactly you want to test for. A challenge we faced was collecting enough test data and the correct test data. As we added new features, our test data didn't test some of the new features which made it quiet difficult to provide useful testing for them

Finally, an important lesson we learned regarding working on a team hardware project remotely was that communication is key to a successful project. Spend the first week or so establishing a system of communicating work done on

a daily or weekly basis. This ensures as little merge conflict and duplicated work as possible.

Glossary of Acronyms

Include an alphabetized list of acronyms if you have lots of these included in your document. Otherwise define the acronyms inline.

- AWS - Amazon Web Services
- CPU - Central Processing Unit
- CNN - Convolutional Neural Network
- CV - Computer Vision
- DNN - Deep Neural Network
- EAR - Eye Aspect Ratio
- GPU - Graphics Processing Unit
- SDK - Software Development Kit

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).

[7] Xiangxin Zhu. "Face Detection, Pose Estimation, and Landmark Localization in the Wild". In: *https://vision.ics.uci.edu/papers/ZhuR$_C$VPR$_2$012/ZhuR$_C$VPR$_2$012.pdf* (2012).

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 18: 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 19: A table of the Device Metrics

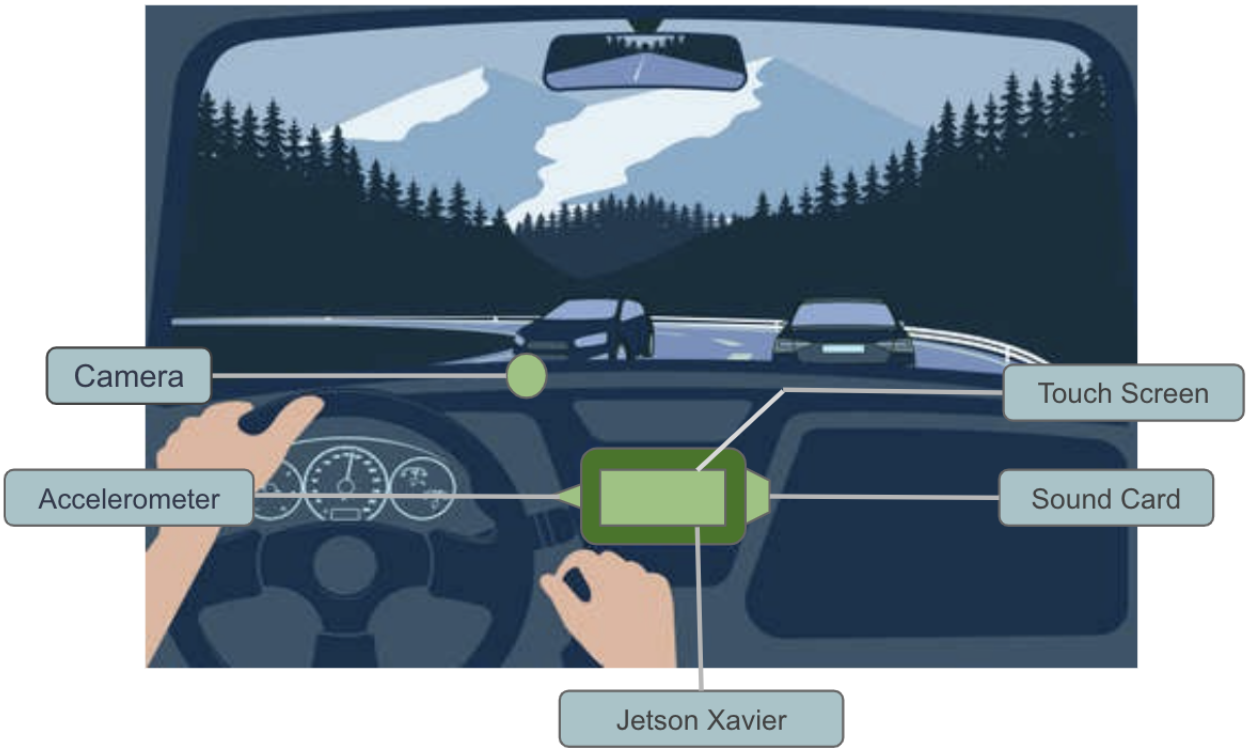


Figure 20: Mock-up of our Device

Figure 21: Mock-up of calibration process

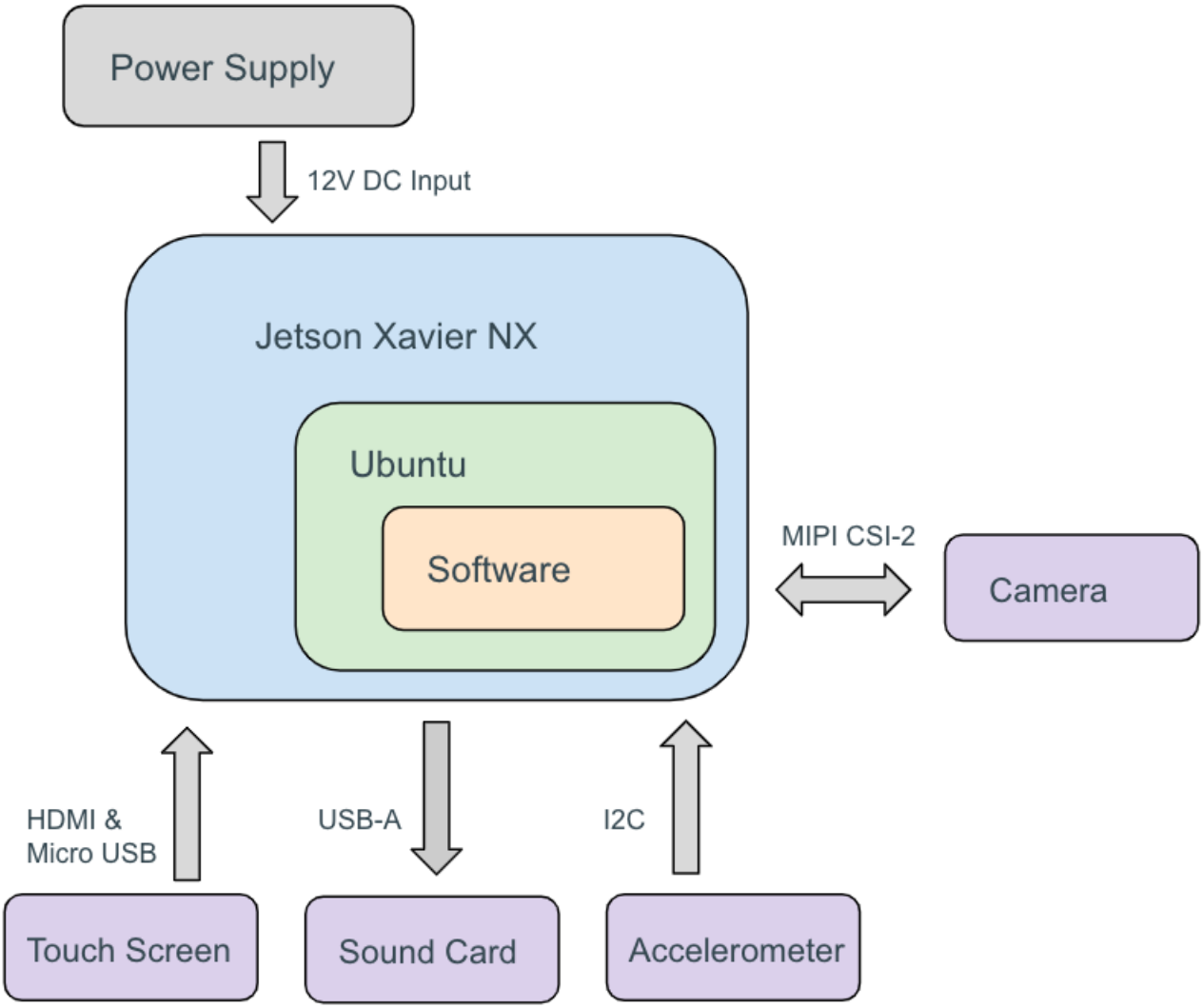


Figure 22: Hardware architecture of CarMa

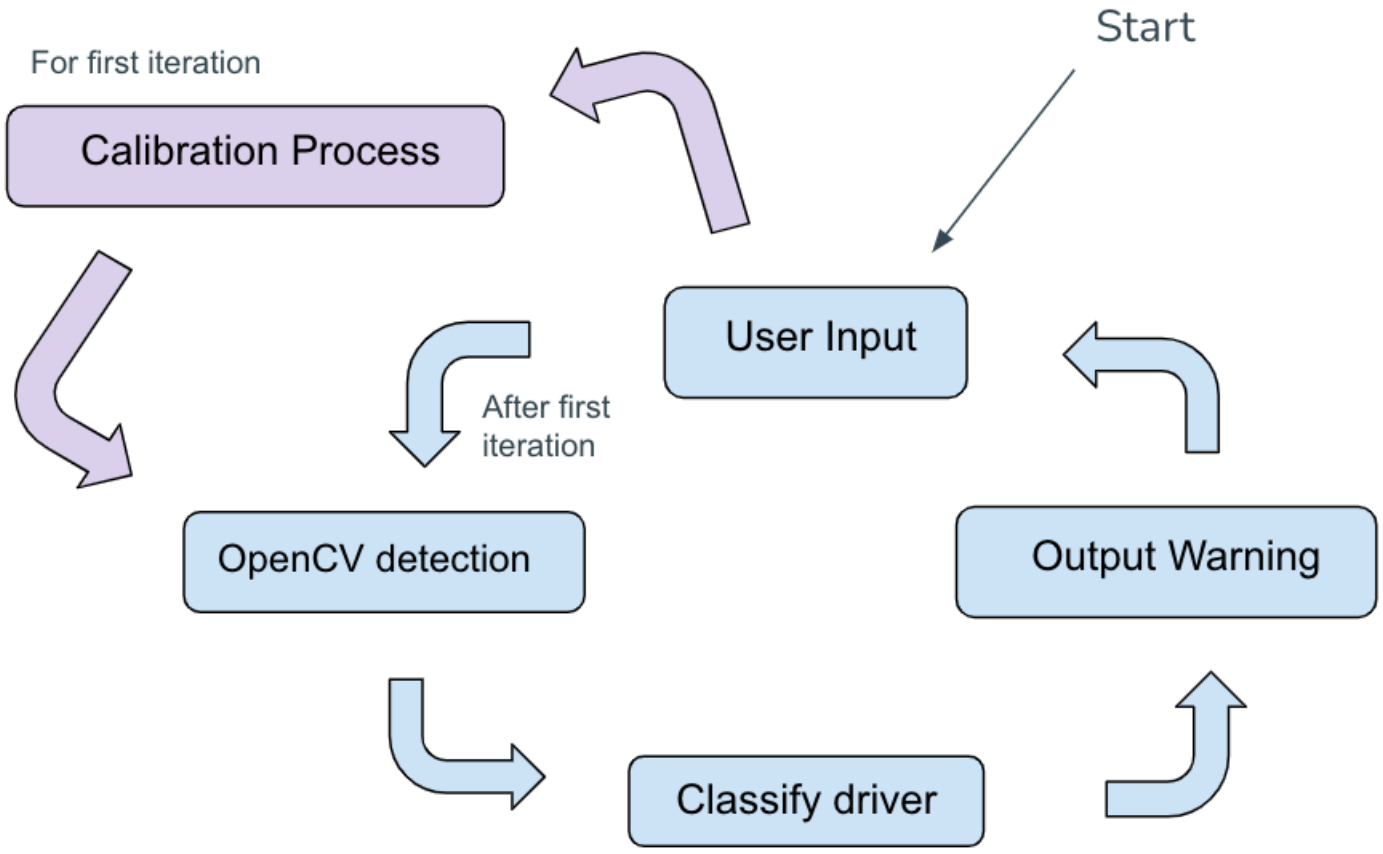


Figure 23: General Software cycle of CarMa

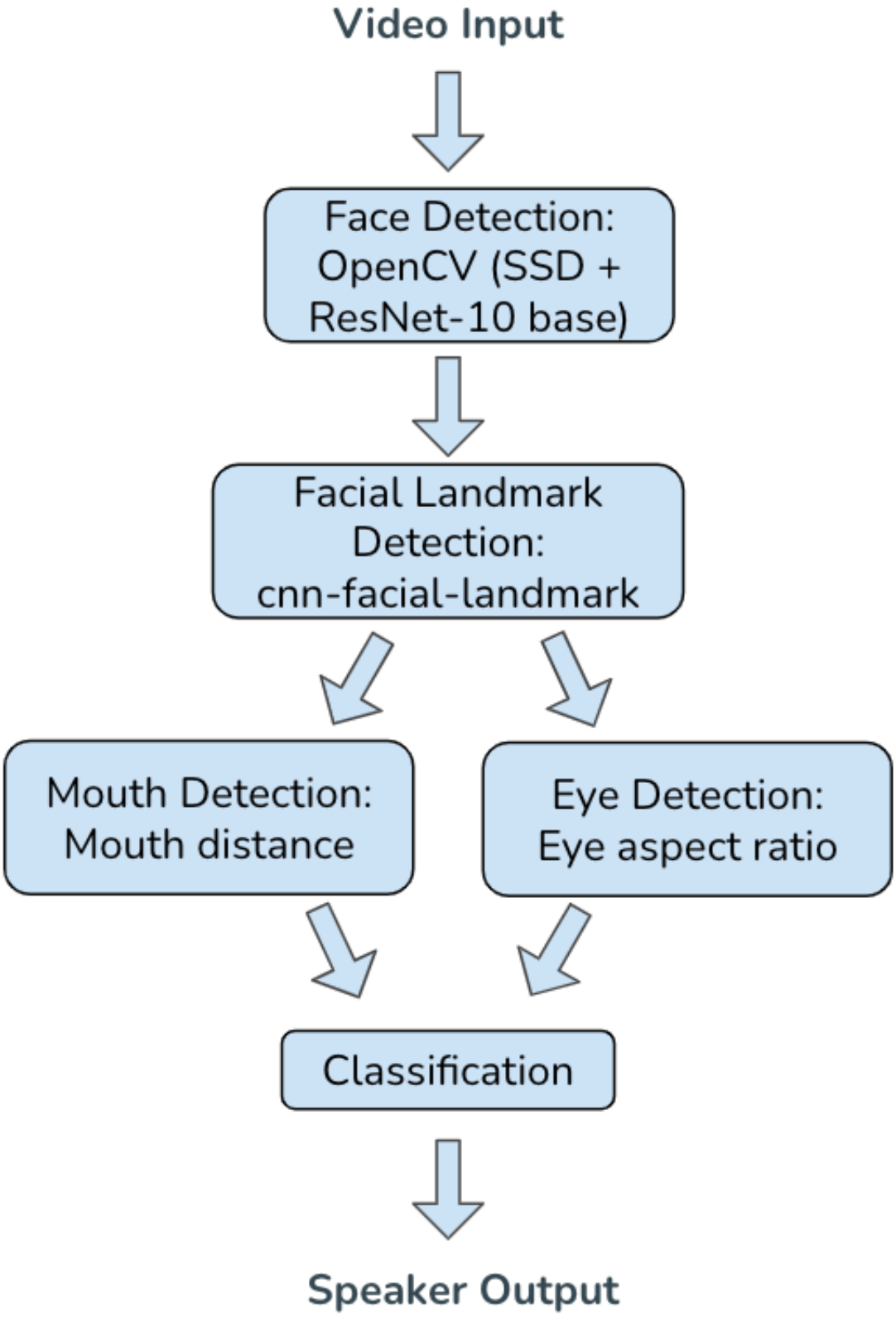


Figure 24: Block Diagram for the Computer Vision Algorithms

Metrics, Validation & Testing

Metrics	Test Plan	Test Results	Status	Prioritize False Positives
Frontal view: detect yawning	Error rate <20%	= 1%		
Frontal view: detect eyes closed	Error rate <30%	= 20.6%		
Frontal view: Eyes looking away	Error rate <40%	= 40%		
Side view: turned > 2 secs	Error rate <35%	Stretch goal		
Latency < 1000ms	Measure the time between action and output	183.3 ms		N/A
>= 5 fps	Measure frame rate with person looking at camera for 60 secs	5.70 fps		N/A
Error Rate of Algorithm	Accuracy of model against test suite >= 75%	85.9%		N/A

Figure 25: Block Diagram for the Computer Vision Algorithms

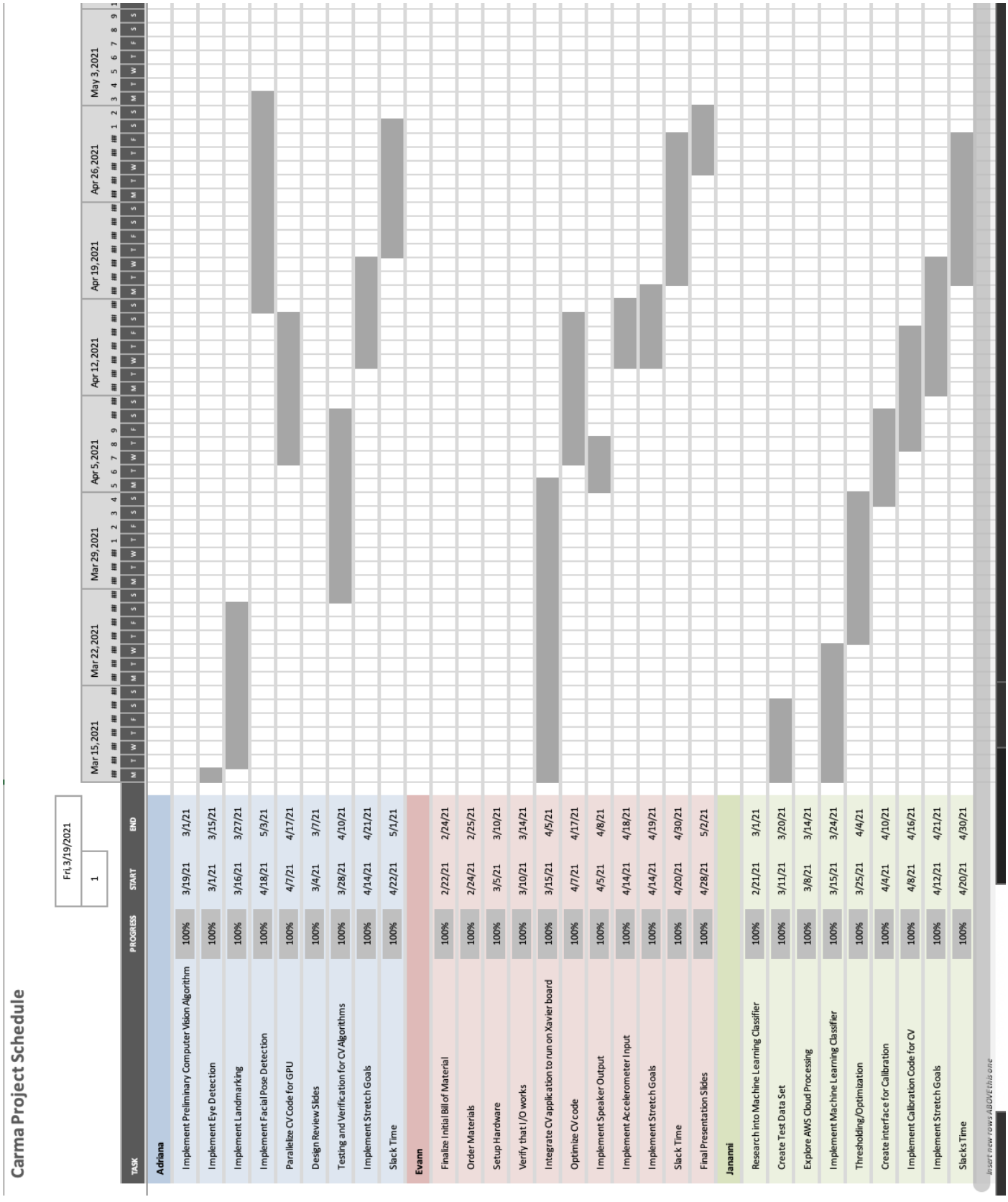


Figure 26: Schedule Gantt Chart