AN INTELLIGENT MOBILE APP TO DETECT DROWSY DRIVING WITH ARTIFICIAL INTELLIGENCE

Thomas Xiao[1] and Yu Sun[2]

[1]Yorba Linda High School, Yorba Linda, CA 92886
[2]California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

Drowsy driving is lethal- 793 died from accidents related to drowsy driving and 91000 accidents related to drowsy driving occurred [1]. However, drowsy driving and accidents related to drowsy driving are preventable. In this paper, we address the problem through an application that uses artificial intelligence to detect the eye openness of the user. The application can detect the eyes of the user via computer vision. Based on the user's eye openness and frequencies, the sleepy driving condition can be inferred by this application. We applied our application to actual driving environments on the highway, both day and night, as well as within a normal control situation using a qualitative evaluation approach. The result shows that it is 88% effective during the day and 75% effective during nighttime. This result reveals effectiveness and accuracy of detection during daytime application under controlled testing, which is more flexible and efficient comparing to previous works. Effectiveness and accuracy for nighttime detection and detections with the presence of other distractions can be further improved.

KEYWORDS

Drowsy driving, Mobile Application, Artificial Intelligence, Driving Safety.

1. INTRODUCTION

Drowsy driving, as simple as it sounds, is driving while sleepy [10, 11, 12]. Drowsy driving is an acute problem. It can occur with any driver of any age group around the world, affecting millions even if they are not behind the wheel. The Centers for Diseases control and Prevention (CDC) once estimated that 1 in 25 adult drivers report having fallen asleep while driving in the previous 30 days [1]. Drowsy Driving often ends in accidents with a variety of effects: car crash, injury, destruction of interstates and roads, and in the worst-case scenarios, death. In fact, the National Highway Transportation Safety Authority estimated that there were 72,000 crashes, 44,000 injuries, and 800 deaths related to drowsy driving in 2013, and some believe even this is underestimated [2]. In California alone, in 2016, 2% of traffic accident deaths were caused by drowsy driving [3]. Each one of these numbers represent human lives, and there are ways to prevent all these avoidable injuries and deaths. Those who are injured badly by car accidents often end up in the ICU, which is even more dangerous during the Coronavirus Pandemic. When people are sleepy behind the wheel, they tend to display certain eye patterns that are recognizable. With a functioning app that can detect and notify the user of drowsy driving, many lives could be saved.

There are not many technological techniques to detect drowsy driving. However, the most advised technique for drivers to prevent drowsy driving is to rest well before driving and stop driving immediately when feeling drowsiness; this method has many flaws since many people do not have enough conscientiousness to first consider rest. For example, most drivers would not want to rest in the middle of a trip. Other examples would be commercial drivers whose job is to drive for most of their days. There are different methods developed by universities, such as the research conducted by H.J Dikkers and M.A. Spaans from Delft University of Technology [6, 13]. Their research conducted on drowsy driving detection depended on facial expression detection. Facial expression detection requires three steps of detection, which are: Facedetection, Facial expression data extraction, and facial expression classification [4]. Although this method can be extremely accurate, it was tested on a computer with an old-style recording camera. The results were accurately collected but have never been projected onto the UI of a mobile application. Their method has never been tested on a modern-day cell phone and is too complex to run in real time on a phone, so is therefore not suitable as a mobile phone app. Also, the algorithm required facial expression detection, which means that facial hair or face coverings on the subject’s face may produce inaccurate or false results, which is extremely unhelpful during Covid-19.

In this paper, we present a new approach to detect and curb drowsy driving. Our goal is to develop an application with an algorithm that would use the detection of eye openness to determine if the driver is driving under drowsy conditions. The method we have developed relies on the collaboration of Google Firebase’s eye detection algorithm. This algorithm was written in Dart language and utilizes the Flutter Camera package plug in to access the phone’s face cam [14, 15]. When the user clicks the start button of the application, the algorithm is programmed to automatically take 10 photos per second. All the pictures taken are analyzed with Google Firebase’s eye openness detection. The Google firebase eye detection can detect the eye openness of a subject and returns a value from 0 to 1 based on how much their eyes are open (0 means closed, while 1 means open). Based on the information returned by Google firebase, we can make calculations of the value returned by Google Firebase to determine if the driver is driving while drowsy. The algorithm determines when to notify the user that they are sleepy through a specifically designed calculation, which will be discussed in the next section. There are some good features present in the algorithm of this current app. First, the algorithm detects eye openness without relying on facial expression. This would be extremely helpful because facial hair and other face coverings may affect the accuracy of results or return a false result to the user. Second, this method has a complex algorithm (a multiple step calculation shown in the third section) to determine if the driver is drowsy when driving. This ensures the data is returned accurately and there are no false alarms [9].

To evaluate the accuracy of our method, we tested the accuracy of the application’s design within different situations that drivers can experience in real life. The factors that we decided to test include Day time without glasses vs with glasses, nighttime without glasses vs with glasses, shade without glass and with glass, location of the phone glass vs without glass. Within each trial, we observed if the detection would trigger a warning. The results of these real-time detection proves to be effective overall.

The rest of the paper is organized as follows: Section Two gives the details on the challenges that we met during the experiment and while designing the sample; Section Three focuses on the details of our solutions corresponding to the challenges mentioned in Section Two; Section Four presents the relevant details of the experiment, followed by the related work in Section Five. Finally, Section Six gives concluding remarks, as well as the future work of this project.

2. CHALLENGES

A few challenges arose while developing this application, as follows.

2.1. Which Part of the Face to Detect

There are multiple parts of our face we can monitor for drowsy driving, such as eyes, mouth, facial expression, or all of them. Each detection has its own benefits and downsides. We can use eyes because if we close our eyes too frequently or our eye openness becomes too small, we can assume the user is tired. We can also use facial expression because when people are tired, they tend to have certain facial expressions. It would be an extremely accurate detection if we could use all the mechanisms concurrently, yet we are developing an application that runs on a cell phone so the algorithm cannot be overly complicated such as could be run on a computer. Each type of detection has its downside, e.g., you cannot determine the drowsiness of a user through eye detection if the driver is wearing sunglasses [4].

2.2. Unprecedented Challenges

An algorithm that can accurately detect drowsy driving is the key to developing a successful application. But there are not many previous works or methods from which we may gain insight. All previously published works involved data analyzed on a computer, not a smartphone. These previous apps were never intended to be an algorithm used by a phone application. We are trying to develop an application that runs on a phone in which the app's algorithm correctly detects and warns the user of drowsy driving, so we are coming up with our own unique algorithm suitable to run on a phone. We were also faced with the challenge of how the app would be able to gather data constantly and automatically from the user through the phone's camera.

2.3. Designing an Effective Algorithm

As stated earlier, the core of the application is an effective and accurate algorithm. There are many different algorithms we can implement for this app. However, since this app is going to be used on real roads by real drivers, there are many unpredictable factors that we need to consider. We also need to determine what is the most effective way to gather data from the user and how to calculate a value to determine if the user is sleepy. We will need an extremely effective and accurate way of collecting data on the user's face movements. We also need to consider the most accurate algorithm to determine when to notify the user.

3. METHODOLOGY

An overview of the system is presented in Figure 1. The user would first have to choose a sound (input) they would like to play for the detection (input by user). Then when the user starts the detection, the phone's face camera would gather the value for each of the eyes with the help of Google firebase eye detection. After gathering the data of the user's eyes, the algorithm would then determine if the value collected is considered closed or open. Then the value would be passed on to two calculations. One for the eye opened, one for the eye closed. The calculated value would then determine when the notification sound would be played. At the same time, a timer also records how long the user has been detecting. While detecting, the user can choose to pause the detection at any time. When the detection is done, the user interface would display the overall status of the user- in this case sleepy or not- and the time they have been using the detection.

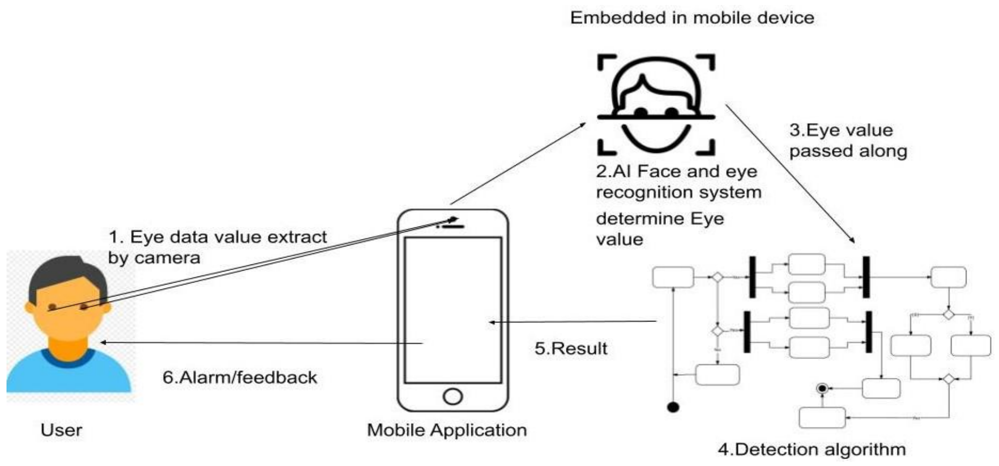


Figure 1. Schematic of system

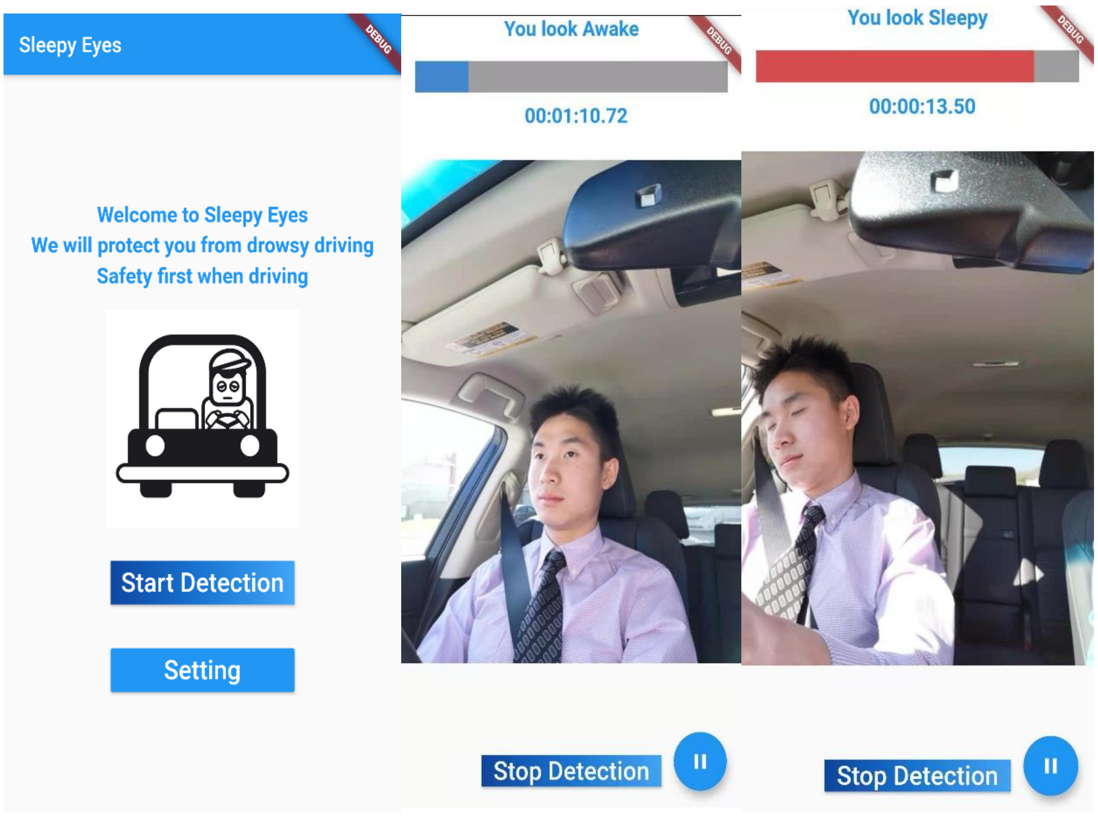
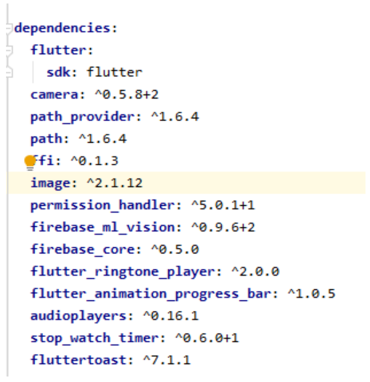


Figure 2. System in use

[image 1] Note the progress bar, first calculation.
[image 2] Note that nothing shows in the progress bar, eyes are open, second calculation is conducted.
[image 3] Progress bar more than halfway, which means "closed_to_total" is greater than 0.5, notification sound is played.

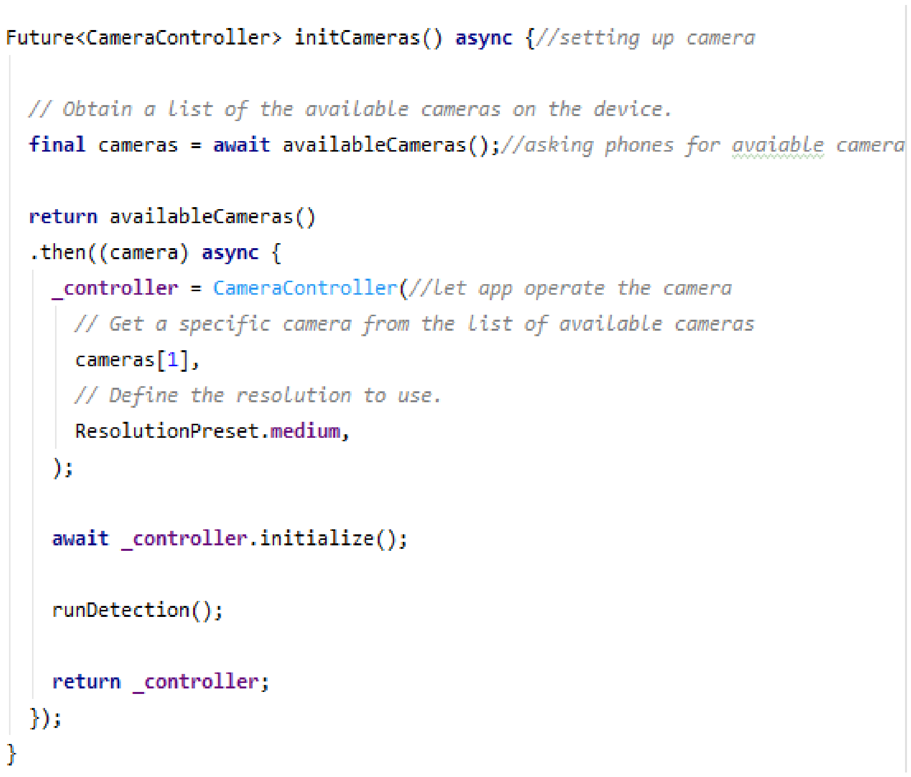
3.1. Camera

To be able to collect the user's eye value, we need to use the user's face cam to get the value. We were able to do this by first implementing a camera package. First, we add camera to the pubspace.yaml file. Then we import the camera package. Finally, we set up the camera, so the camera usage of this app is through the face camera.



```
dependencies:
  flutter:
    sdk: flutter
  camera: ^0.5.8+2
  path_provider: ^1.6.4
  path: ^1.6.4
  ffi: ^0.1.3
  image: ^2.1.12
  permission_handler: ^5.0.1+1
  firebase_ml_vision: ^0.9.6+2
  firebase_core: ^0.5.0
  flutter_ringtone_player: ^2.0.0
  flutter_animation_progress_bar: ^1.0.5
  audioplayers: ^0.16.1
  stop_watch_timer: ^0.6.0+1
  fluttertoast: ^7.1.1
```

Figure 3. Set up



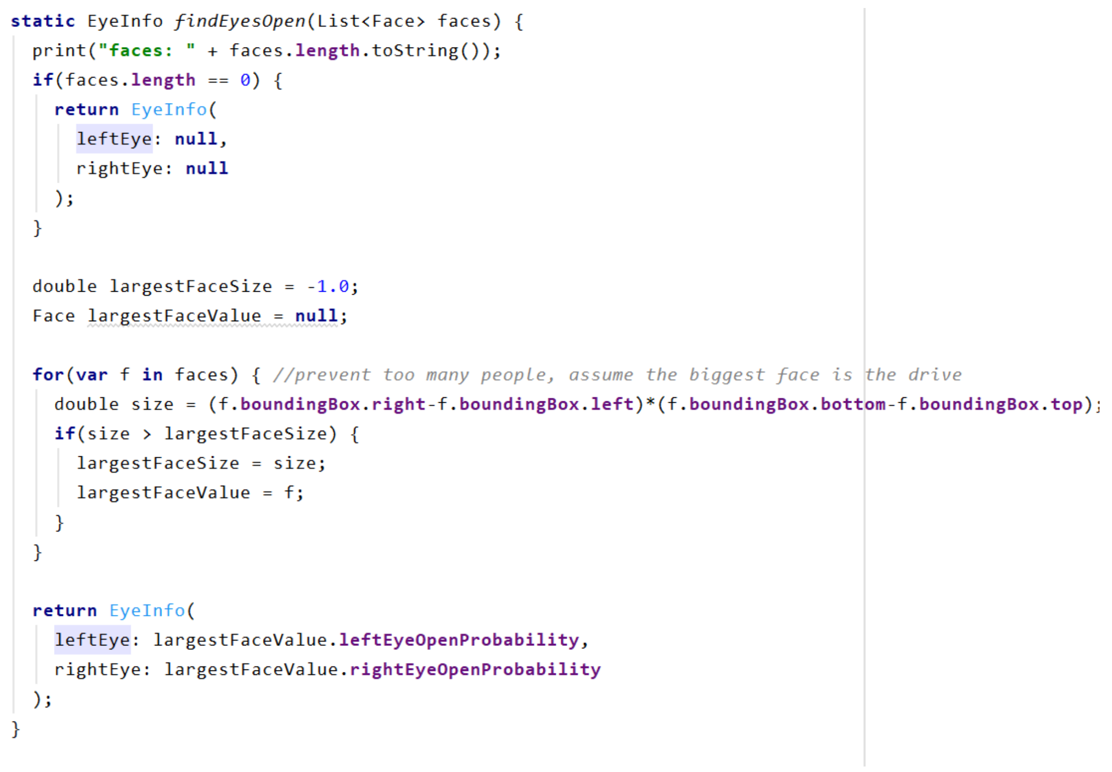
```
Future<CameraController> initCameras() async {//setting up camera

  // Obtain a list of the available cameras on the device.
  final cameras = await availableCameras();//asking phones for avaiable camera

  return availableCameras()
  .then((camera) async {
    _controller = CameraController(//let app operate the camera
      // Get a specific camera from the list of available cameras
      cameras[1],
      // Define the resolution to use.
      ResolutionPreset.medium,
    );

    await _controller.initialize();

    runDetection();

    return _controller;
  });
}
```

Figure 4. Set up code

3.2. GoogleFirebase (ML core)

We implemented Google firebase face detection. The face detection can return values for the user’s eyes positions and eye openness. For this application, we need the data collection of the eye openness value. We allow the app to retract eye openness values of the user by adding

Google-service.json file to the app's source (src) folder, which would enable us to use its face recognition service. We need another class file specifically in the lib folder along with other codes in order for the Google-service.json file to work perfectly alongside of other aspects of the app and to implement a face detector. After adding the Google-service.json file, we were able to utilize and design an AI eye detection system with the help of basic Google firebase face recognition system. The picture below is how we utilized the face detection code originally given and added in some of our codes to retrieve the eye value. With this code, the app can work perfectly with other parts of the program:



```
static EyeInfo findEyesOpen(List<Face> faces) {
  print("faces: " + faces.length.toString());
  if(faces.length == 0) {
    return EyeInfo(
      leftEye: null,
      rightEye: null
    );
  }

  double largestFaceSize = -1.0;
  Face largestFaceValue = null;

  for(var f in faces) { //prevent too many people, assume the biggest face is the drive
    double size = (f.boundingBox.right-f.boundingBox.left)*(f.boundingBox.bottom-f.boundingBox.top);
    if(size > largestFaceSize) {
      largestFaceSize = size;
      largestFaceValue = f;
    }
  }

  return EyeInfo(
    leftEye: largestFaceValue.leftEyeOpenProbability,
    rightEye: largestFaceValue.rightEyeOpenProbability
  );
}
```

Figure 5. Adding code to retrieve eye values

The AI face detector will be used alongside later in many places. The image above is the basic codes of the face detector.

3.3. Detection Algorithm

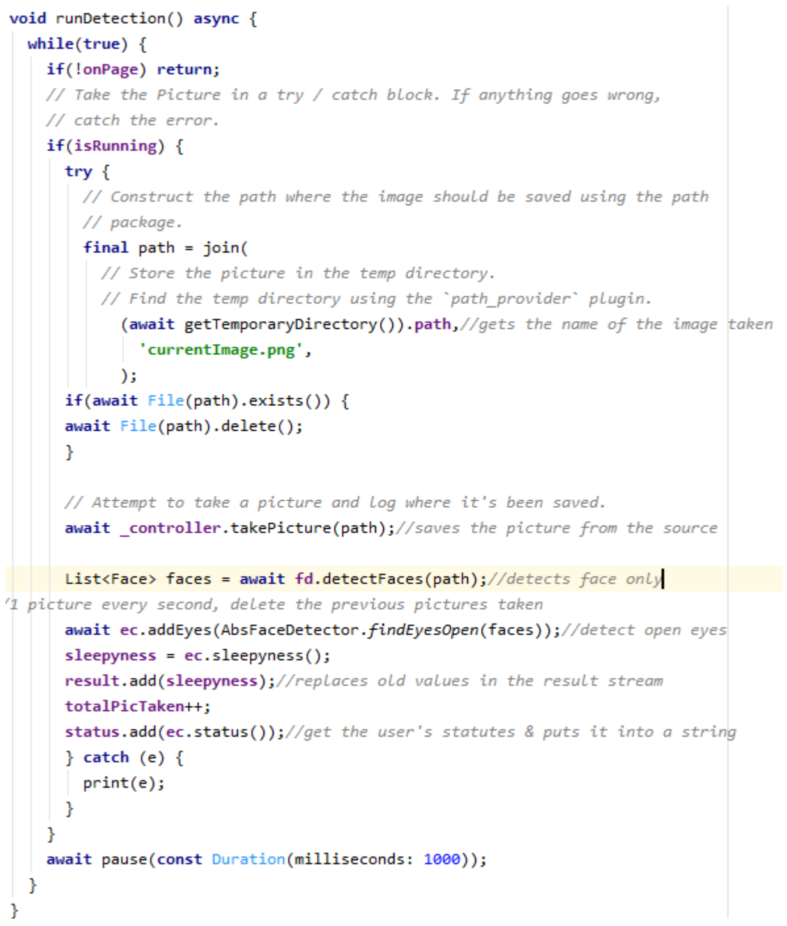
The most important part of this app is the detection algorithm. We show all the logic gates that would trigger the system to notify the user being sleepy. The previous two sections (Camera and Google Firebase) are just setting up the basics for this detection. This part is when we put everything together. To be able to run the detection well, we need to first set the camera to be able to continuously take pictures and for the Google firebase to detect the face. The camera is automatically set to take one picture every minute during detection. The detection begins with the algorithm taking one image of the user. If nothing goes wrong with taking the image, the algorithm will go on and construct a path to store the image in a temporary directory and then use the path-provider plugin (plugin for finding commonly used locations on the filesystem) to locate it [5]. Then, the new image path taken would be stored and the old image path deleted. After the image has been temporarily stored in the path, the face detector mentioned previously would determine if there are any faces. If there are, then it would retrieve the user's eye value. Then, the eye value retrieved would be entered into the following calculation. We defined any value below 0.3 as sleepy. If the eye value is smaller than 0.3, a variable named “closed_to_total” starting with an initial value of 0 would be going through this formula:

```
closed_to_total = closed_to_total*(3/4) + 1/4;
```

The calculation is done this way because we would like the progress bar (shown below) to increase not linearly or exponentially. This is designed to quickly notify the user if the detection detects that he/she is sleepy. If the retrieved eye value is greater than 0.3, then the following calculation would be conducted:

```
closed_to_total = closed_to_total*(15/16);
```

This would decrease the progress bar much slower than when the bar is increasing. It is done this way to let the user go to rest quickly and let the notification sound keep playing. The “closed_to_total”variable determines when the app would play a notification sound. If the “closed_to_total”value is greater than 0.5, then the notification sound would be played. The sound would keep playing(as a loop) unless closed_to_total is smaller than 0.5. The image below shows the user interfaces when the first calculation is conducted and when the second calculation is conducted. It also shows when the notification sound is played.



```
void runDetection() async {
  while(true) {
    if(!onPage) return;
    // Take the Picture in a try / catch block. If anything goes wrong,
    // catch the error.
    if(isRunning) {
      try {
        // Construct the path where the image should be saved using the path
        // package.
        final path = join(
          // Store the picture in the temp directory.
          // Find the temp directory using the `path_provider` plugin.
            (await getTemporaryDirectory()).path,//gets the name of the image taken
            'currentImage.png',
          );
      if(await File(path).exists()) {
      await File(path).delete();
      }

      // Attempt to take a picture and log where it's been saved.
      await _controller.takePicture(path);//saves the picture from the source

      List<Face> faces = await fd.detectFaces(path);//detects face only
'1 picture every second, delete the previous pictures taken
      await ec.addEyes(AbsFaceDetector.findEyesOpen(faces));//detect open eyes
      sleepyness = ec.sleepyness();
      result.add(sleepyness);//replaces old values in the result stream
      totalPicTaken++;
      status.add(ec.status());//get the user's statutes & puts it into a string
      } catch (e) {
        print(e);
      }
    }
    await pause(const Duration(milliseconds: 1000));
  }
}
```

Figure 6. code for “closed_to_total” values and sound notification

3.4. AI and algorithm integration

As shown above, we used artificial intelligence eye and face recognition to first detect the face currently in the camera's frame and the eye values on the face. After the AI face recognition retrieves the eye's data, we use these data and put them through the calculation and decide when to notify the user of being sleepy.

4. EXPERIMENT

To evaluate the accuracy of our method, we decided to conduct experiments in real-life driving situations. We decided to test the accuracy of this application's design in many situations that drivers can experience in real life. The factors that we decided to experiment include Day time Without Glasses vs Glasses, Nighttime Without Glasses vs Glasses, Shade Without glass and with glass, location of the phone glass vs without glass. For the first three experiments, the phone is set up at 70 cm from the tester (just as the image below) while for experiment four, the phone is set up at 87cm from the user. Within each trial, we are going to see if the detection would trigger a warning. The accuracy of each detection would be determined as follow: if the test subject's eyes are closed and no alarms have sounded, then the accuracy is 0%. But if the test subject's eyes are open and the alarm goes off, then the accuracy level is 100%. For example, if the detection is done accurately three times out of the four times, it would be a 75% accuracy. The following is the result experiment accuracy data is as follows: 88% effective during day time and 75% effective during night time. Here is a picture of the experiment set up:

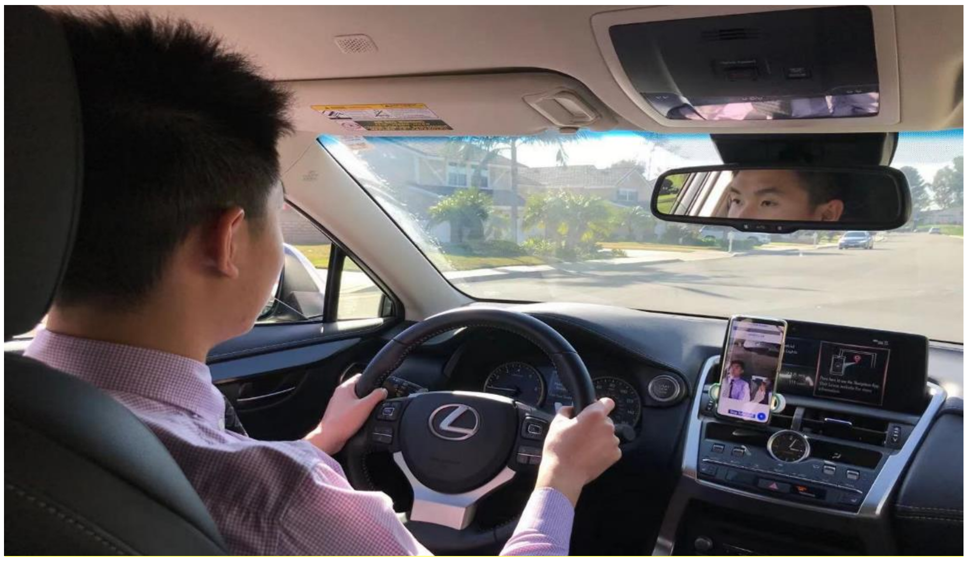
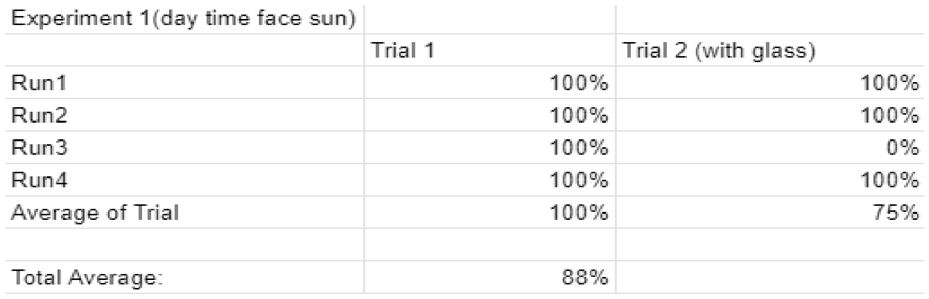


Figure 7. Experiment set up

4.1. Experiment on Daytime Facing Sun Without Glasses Vs With Glasses

For this experiment, the glass used in the experiment is an ordinary correctional lens. We are facing the sun during this experiment and sitting in the car. We decide the accuracy of the algorithm through a simple test: if I close my eyes and no alarms, then the accuracy is 0%. But if I close my eyes and the alarm goes off, then the accuracy level is 100%. The following is the result experiment accuracy data is as follow:



Experiment 1(day time face sun)		
	Trial 1	Trial 2 (with glass)
Run1	100%	100%
Run2	100%	100%
Run3	100%	0%
Run4	100%	100%
Average of Trial	100%	75%
Total Average:	88%	

Figure 8. Data table for Experiment 1

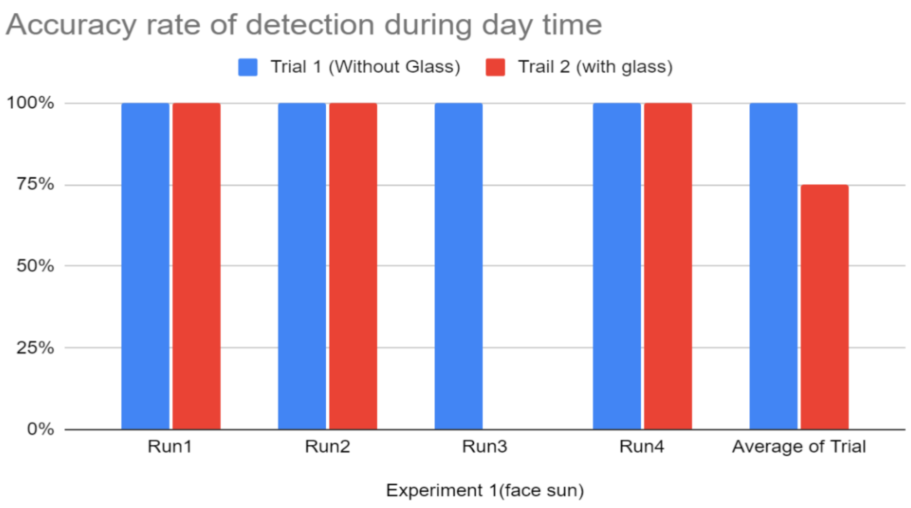
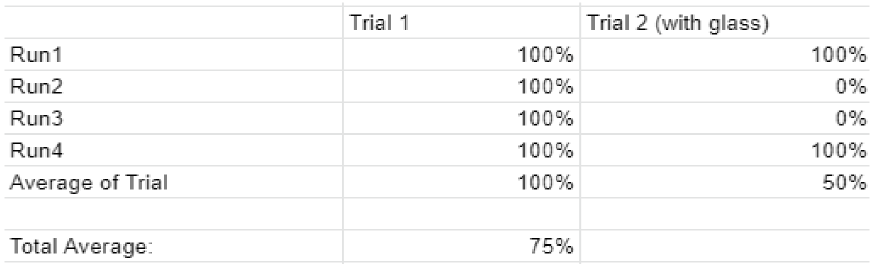


Figure 9. Accuracy rate of detection: Daytime

Based on the results, we can see this algorithm has a 100% accuracy when detecting during daytime with no glasses and a 75% accuracy rate with glasses. The overall accuracy rate for daytime detection when facing the sun is 88%.

4.2.Experiment on night without Glasses vs Glasses

For this experiment, the glass used in the experiment is an ordinary correctional lens. We conducted this experiment during nighttime and sitting in the car. We decide the accuracy of the algorithm through a simple test: if I close my eyes and no alarms have sounded, then the accuracy is 0%. But if I close my eyes and the alarm goes off, then the accuracy level is 100%. The following is the result experiment accuracy data is as follow:



	Trial 1	Trial 2 (with glass)
Run1	100%	100%
Run2	100%	0%
Run3	100%	0%
Run4	100%	100%
Average of Trial	100%	50%
Total Average:	75%	

Figure 10. Data table for Experiment 2

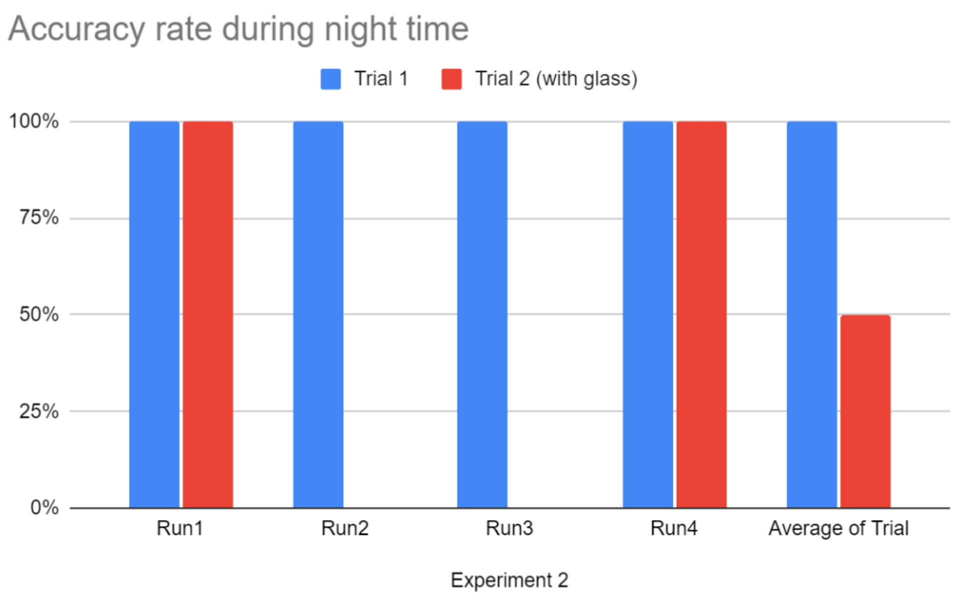
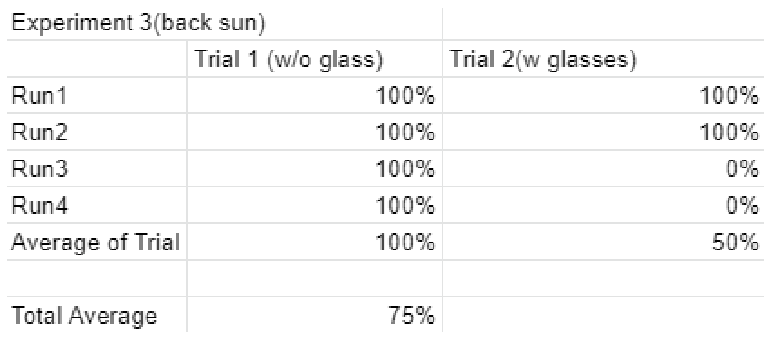


Figure 11. Bar chart of nighttime accuracy

Based on the results, we can see this algorithm at night has a 100% accuracy with no glasses and a 50% accuracy rate with glasses. The overall accuracy rate at night is 75%.

4.3. Experiment on Sunshade with and Without Glass

For this experiment, the glass used in the experiment is an ordinary correctional lens. Our backs are facing the sun during this experiment and sitting in the car. We decide the accuracy of the algorithm through a simple test: if I close my eyes and no alarms have sounded, then the accuracy is 0%. But if I close my eyes and the alarm goes off, then the accuracy level is 100%. The following is the resulting experiment accuracy data:



Experiment 3(back sun)		
	Trial 1 (w/o glass)	Trial 2(w glasses)
Run1	100%	100%
Run2	100%	100%
Run3	100%	0%
Run4	100%	0%
Average of Trial	100%	50%
Total Average	75%	

Figure 12. Data table for Experiment 3

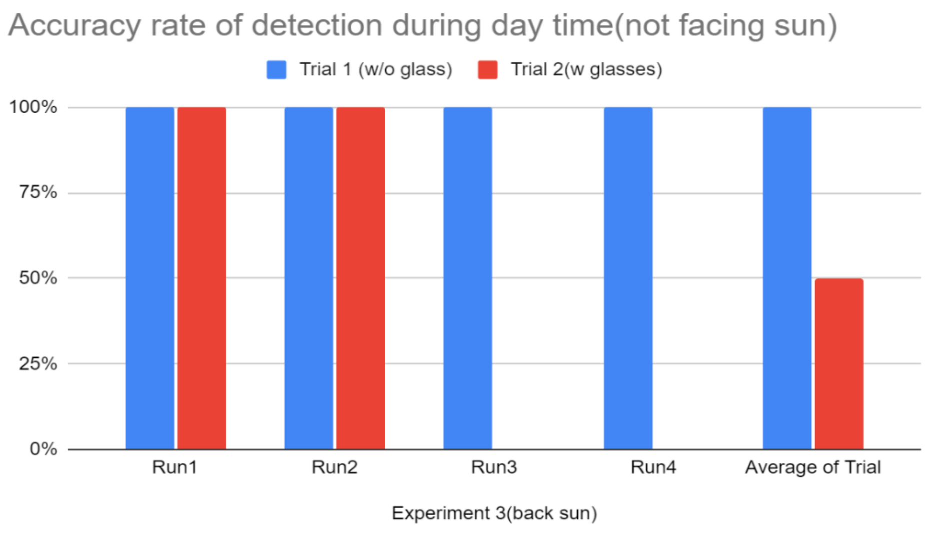
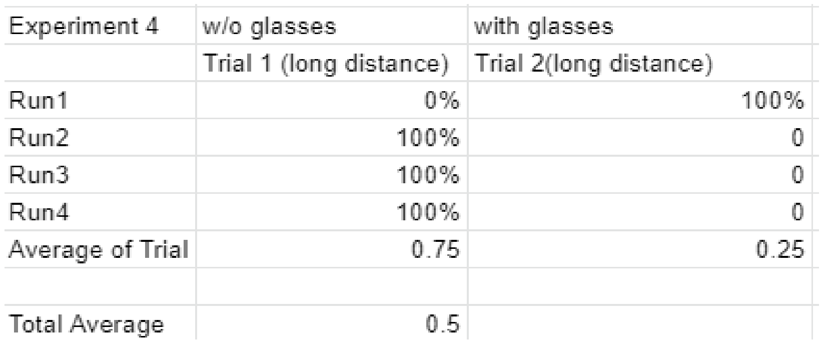


Figure 13. Bar chart of daytime accuracy (not facing sun)

Based on the results, we can see the algorithm has a 100% accuracy when detecting during the daytime with no glasses and a 50% accuracy rate with glasses. The overall accuracy rate for daytime detection when our back is facing the sun is 75%.

4.4.Experiment on Phone at a Further Distance and Glasses

For this experiment, we have decided to place the phone 87 cm from the user's eyes. (In the previous three experiments, the phone was placed 70cm from the user.) This experiment is conducted under daytime conditions with the sun facing from the back and with variables of wearing and not wearing glasses. The experiment is conducted in a car. We decide the accuracy of the algorithm through a simple test: if I close my eyes and no alarms have sounded, then the accuracy is 0%. But if I close my eyes and the alarm goes off, then the accuracy level is 100%. The following is the resulting experiment accuracy data:



Experiment 4	w/o glasses	with glasses
	Trial 1 (long distance)	Trial 2(long distance)
Run1	0%	100%
Run2	100%	0
Run3	100%	0
Run4	100%	0
Average of Trial	0.75	0.25
Total Average	0.5	

Figure14. Data table for Experiment 4

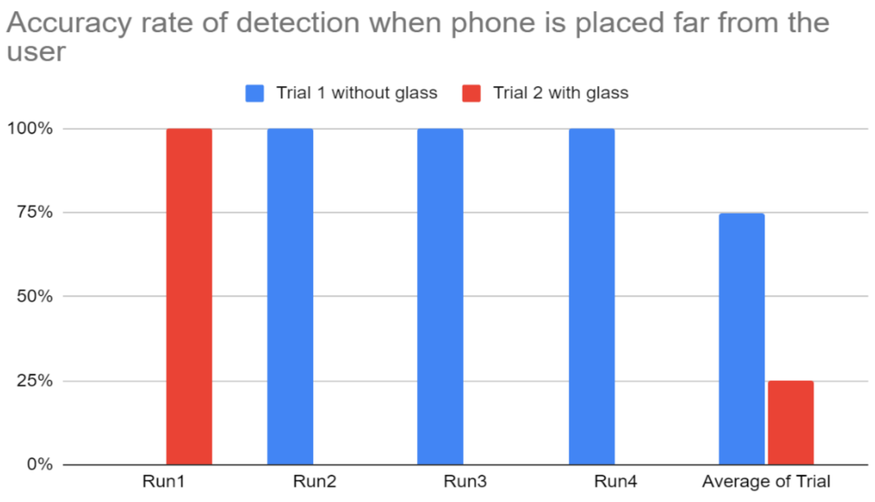


Figure 15. Bar chart of daytime accuracy (phone 87cm from user)

Based on the results, we can see the algorithm has a 75% accuracy when placed at 87 cm from the eyes with no glasses and a 25% accuracy rate with glasses under the same conditions. The overall accuracy rate for daytime detection when the phone is placed at 87cm is 50%.

In conclusion, our method proves to be effective overall. But under certain conditions, the algorithm is not effective. We can improve the algorithm's accuracy under certain conditions, especially nighttime detection. One of the uncontrollable factors is light reflection on the user's glasses. Another uncontrollable variable that can improve the efficiency is setting the phone closer to the user; this is up to the user. In general, our approach to solving this problem is proven to be effective within different experimental conditions.

5. RELATED WORK

Dikkers, et al. present using Face Recognition system for Driver Vigilance Monitoring. Dikkers, et al. use facial expression to detect drowsy driving. They use many similar methods as ours, including data extraction. One aspect that was different was that they tested their algorithm based on facial expressions. Their methods have not been tested in real-life situations and haven't proved to be runnable on mobile applications [6].

Assari, M.A. and M. Rahmati present using infrared lights to first clear out the visual interference such as darkness or light reflections and use facial expression detection algorithms on the face in the video frame. This method of using infrared light is smartly done. They have also tested their model in real-life situations [7].

Xu, L. et al. presented a solution using the percentage of eyelid closure to detect drowsy driving. They relied on factors including blink time and blink rate. The methodology contains many other factors compared to ours and employs different calculations to determine what is considered sleepy. Their methodologies have been tested in real-life situations and on a mobile application [8].

6. CONCLUSION AND FUTURE WORK

In conclusion, we were looking for a solution for drowsy driving through an app that uses an algorithm that can accurately and effectively detect drowsy driving or sleepiness based on the user's eyes that is optimizable on a modern-day smartphone. We decided to use Google firebase's face recognition algorithm and tweak it to extract the eye data from the user. We would then use the eye detector algorithm we developed to extract data from the user's eye. When the value is returned, it is passed down into a chain of calculations to determine if the alarm should go off. We applied this method to four different experiments that are closely related to four scenarios drivers would most likely encounter in real-life driving situations. The experiment results show promising accuracy for most conditions but there are also places for improvement. Therefore, our model can apply to everyday life and benefit a lot of people. Anyone from any age group can use our solution to prevent drowsy driving. In the future, we would improve the accuracy for detection when wearing glasses. The method stated in the paper can also be applied in interdisciplinary studies relating to drowsy driving, including behavioral science. Other scientists can utilize this model to further improve the current method of drowsy driving detection.

REFERENCES

[1] https://www.nhtsa.gov/risky-driving/drowsy-driving

[2] https://www.cdc.gov/sleep/features/drowsy-driving.html

[3] https://sci-hub.se/https://ieeexplore.ieee.org/abstract/document/6785367

[4] https://ieeexplore.ieee.org/abstract/document/1400934

[5] https://pub.dev/packages/path_provider

[6] Dikkers, H. J., et al. "Facial recognition system for driver vigilance monitoring." *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*. Vol. 4. IEEE, 2004.

[7] Assari, Mohammad Amin, and Mohammad Rahmati. "Driver drowsiness detection using face expression recognition." *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. IEEE, 2011.

[8] Xu, Lunbo, et al. "Sober-Drive: A smartphone-assisted drowsy driving detection system." *2014 International conference on computing, networking and communications (ICNC)*. IEEE, 2014.

[9] Pandit, Prajwal, et al. "VIVIFY: DRIVER'S DROWSINESS DETECTION AND ALARMING SYSTEM." *International Journal of Advanced Research in Computer Science* 11 (2020).

[10] Vanlaar, Ward, et al. "Fatigued and drowsy driving: A survey of attitudes, opinions and behaviors." *Journal of safety research*39.3 (2008): 303-309.

[11] Stutts, Jane C., Jean W. Wilkins, and Bradley V. Vaughn. "Why do people have drowsy driving crashes." *Input from drivers who just did* 202.638 (1999): 5944.

[12] Tefft, Brian C. "Asleep at the wheel: The prevalence and impact of drowsy driving." (2010).

[13] Spaans, M. A., and H. J. Dikkers. "Facial recognition system for driver vigilance monitoring." *Res. Rep. No. LSS* 169.03 (2003).

[14] Bracha, Gilad. *The Dart programming language*. Addison-Wesley Professional, 2015.

[15] Mikolaj, Miroslav. "Using Flutter framework in multi-platform application implementation."