California State University, Fresno

Lyles College of Engineering

Electrical and Computer Engineering Department

TECHNICAL REPORT

Experiment Title: ____Facial Recognition and Position Tracking Embedded System_____

Course Title: _____ECE 186B: Senior Design II_____

Date Submitted: _____May 12, 2020_____

Advisor: _____Dr. Nan Wang_____

Prepared by:	Sections Written:
_____Jason Luc_____	_____Jason Luc_____
_____Brian Cardwell_____	_____Brian Cardwell_____

INSTRUCTOR SECTION

Comments: _____

_____

_____

_____

Final Grade: Jason Luc: _____

Brian Cardwell: _____

Table of Contents

1. Problem Statement and Objective

1.1 Problem Statement

Distracted driving accounts for approximately 9 percent of fatal car crashes in the United States [1] and drowsy drivers account for an additional 2.5 percent [2]. On average, this accounts for 4,000 lives lost each year in the U.S. alone. Our project will attempt to drastically reduce this number by alerting distracted drivers to help them maintain focus. In addition, this project will help prevent vehicle theft. In 2008, there were about 750,000 reported incidents of vehicle theft in the United States alone[9]. Our project seeks to reduce these two vehicular crimes by implementing features that will track and discourage distracted driving, and one that won't allow the vehicle to start without scanning the face of a person who is registered to the car. This project will incorporate the use of machine learning and inputs from various hardware to accomplish our goal.

1.2 Objective

The project objectives are to build an embedded system that will recognize a registered face and track the position of the face in real time. The Face Recognition process will include the following functionalities:

1. Detect if a face is present.
2. Determine unique features of the face.
3. Compare features between a live capture and a stored copy to determine if they match.

Position tracking will be determined using the following process:

1. Detect if a face is present.
2. Determine unique features of the face.
3. Based on the nose and chin position, determine if the face is drooping downward.

This system could then be used as an add-on to a vehicle's operating system. It would prevent the vehicle from operating if a known user isn't identified. Additionally, it would alert a driver who is looking down while the vehicle is in motion.

2. Background

Machine learning is the science of developing computers to learn and behave as if they were humans. A more detailed explanation is that computers are given a decision algorithm and then fed large amounts of data. As the computer classifies the data using the algorithm the computer's decisions are checked and corrected to strengthen the decision algorithm. This process continues until the computer's decision accuracy is at a level that is deemed suitable by the user. This means that through machine learning computers are able to learn and improve by providing them with information. The information fed to computers are most commonly in the form of observations and real-world interactions. This is a simple explanation of how machine learning works, but there are more things that happen behind the scenes of machine learning.

There are a variety of different machine learning algorithms which are grouped by their learning style or by their form/function. A few examples of each are: supervised learning, unsupervised learning, regression, decision tree, clustering, and deep learning. However, all machine learning algorithms share three components: representation, evaluation, and optimization. Representation is a set of classifiers that the computer is able to understand. Some concepts of representation include K-nearest neighbor, logistic regression, decision trees, neural networks, and Bayesian networks.

Evaluation is an objective or scoring function that allows the computer to assign values or weights to each option. A few concepts that computers use to evaluate which decision gets what weight value are precision and recoil, squared error, posterior probability, K-L divergence, and cost and utility. The final component, optimization, is the search method that computers use to make decisions based on the values of the weights assigned by the evaluation component.

Face Recognition is a Machine Learning technique that combines several different Machine Learning algorithms to perform a single task. The general order of operations for recognizing a particular face is:

1. Locate a face or multiple faces in an image.
2. Find the unique features of the face (eyes, nose, mouth , etc.).
3. Compare these features to known images.
4. Decide whether the features are similar enough to be a match.

3. Technical Plan

3.1 Hardware

The embedded system will be implemented on the NVIDIA Jetson Nano, which is designed for Artificial Intelligence and Machine Learning embedded applications. The Jetson Nano, shown in **Figure 1**, hosts a quad-core ARM processor, 128-core Maxwell GPU, and 4GB of RAM.

Figure 1: Jetson Nano

The Nano contains the following communication ports and protocols (see **Figure 2**):

1. USB 3.0 Type A (4x)
2. Micro USB
3. HDMI
4. Camera Ribbon Connector
5. Ethernet
6. GPIO headers
7. I2C
8. SPI
9. UART

Additional hardware will include a relay, switch, and speaker. They will connect to the GPIO expansion header. The relay will be energized when the face recognition program finds a positive match. The switch is used to represent the state of the vehicle as either moving forward or not moving forward. The speaker will sound if the system is in a moving forward state and the users head is drooping down.

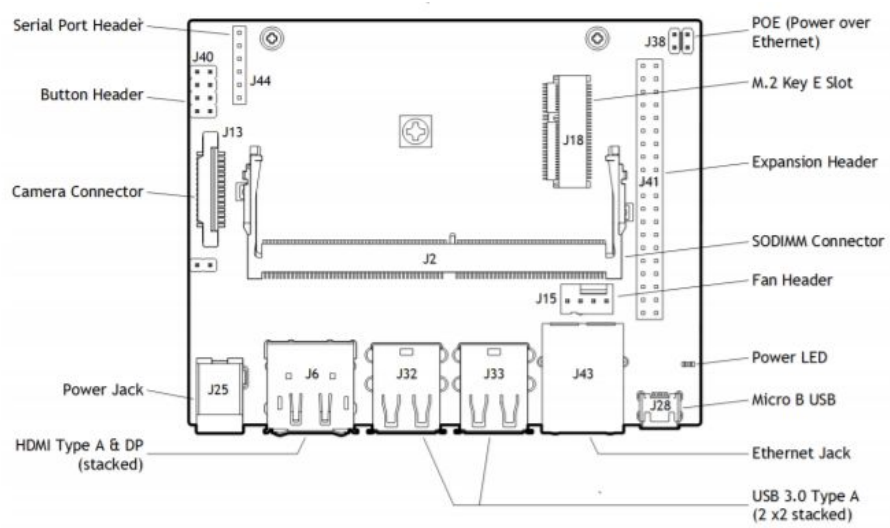


Figure 2: Jetson Nano Connections

3.2 Software

There are several Machine Learning models already developed that can perform Face Recognition with very high accuracy. The goal of the project is to implement these models for specific tasks; not to improve them or develop new models. A high level Python API called *face_recognition* will be used when creating the software for the project [5]. The API is built on an open-source Machine Learning library called *dlib*. The Face Recognition model developed by dlib has an accuracy of 99.38 percent according to the benchmark called Labeled Faces in the Wild (LFW) [6]. The dlib model is designed using a deep convolution neural network (D-CNN) based on a ResNet-34 network developed by Microsoft [7].

The first step in the process is locating faces within an image. To accomplish this, a technique called Histogram of Oriented Gradients (HOG) is used [8]. A HOG classifier looks at each pixel in an image and compares it to all neighboring pixels. Then it creates a gradient in the direction which pixels are becoming darker, see **Figure 3**. The magnitude of the gradient is proportional to the intensity of the change in darkness. A face can then be identified by comparing it to the gradient of a pre-trained model of millions of faces, see **Figure 4**.

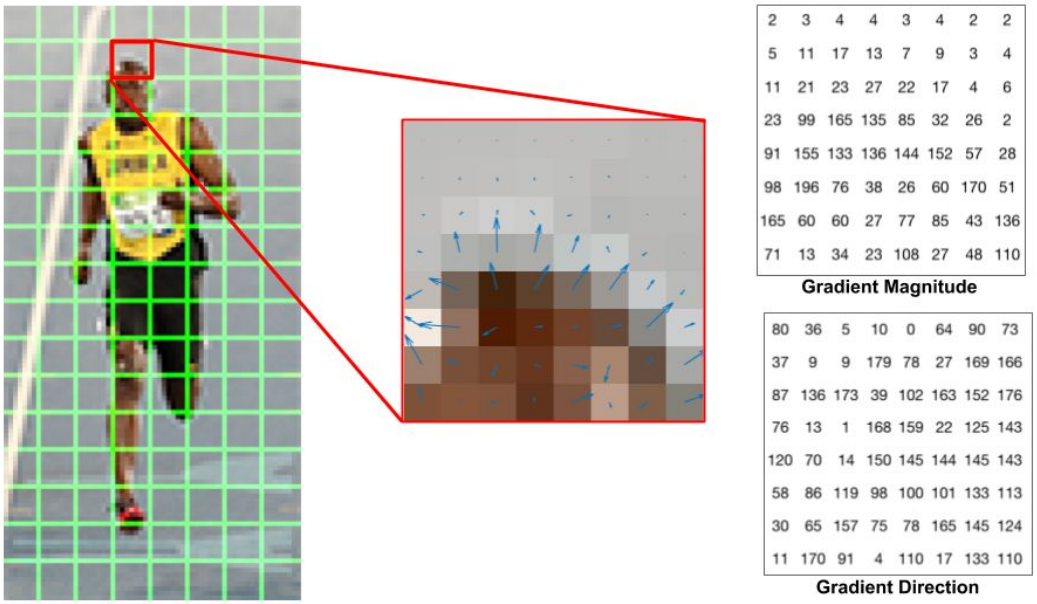


Figure 3: HOG Gradients

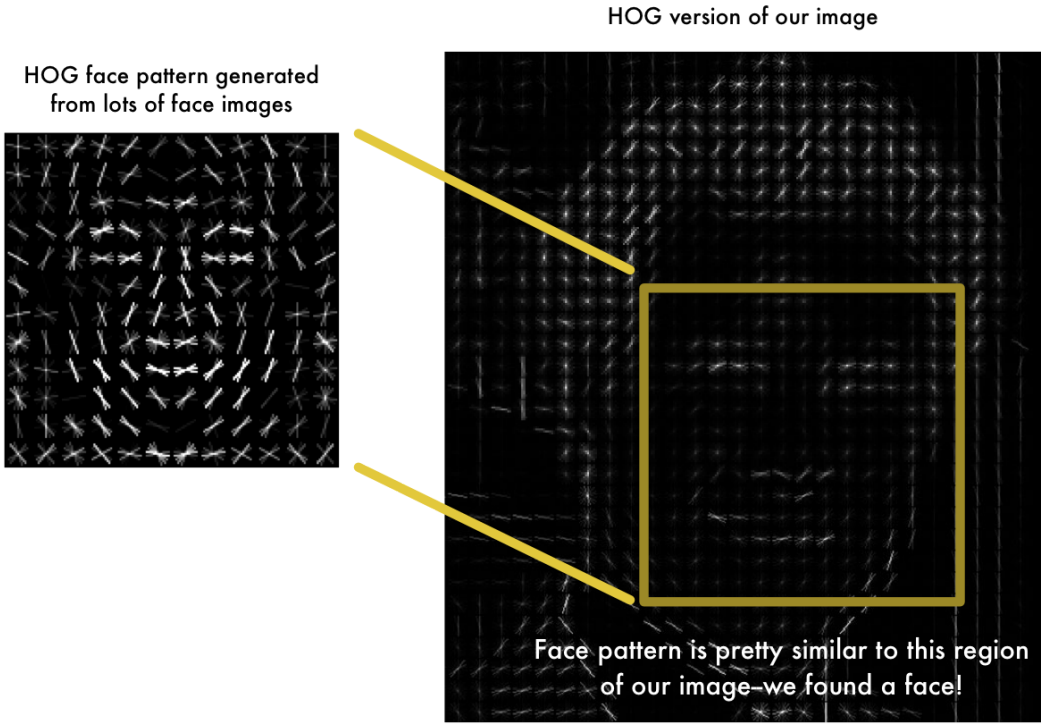


Figure 4: HOG Detector

The next step is determining unique features of the face. This is accomplished using a regression tree algorithm to detect 68 specific face landmarks[9,10], as shown in **Figure 5**.

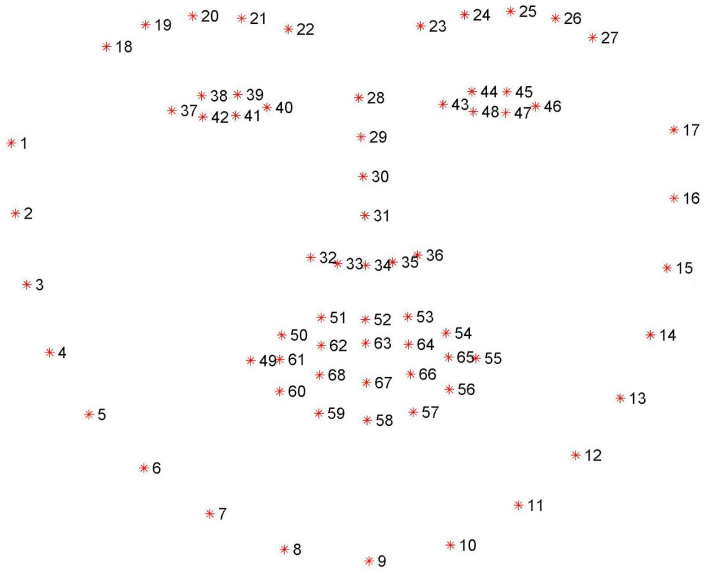


Figure 5: Face Landmarks

Landmarks are necessary because it acts as reference points so that a face can be turned in any direction and still be recognized. Once landmarks are identified the face is given a 128 dimension byte vector using a D-CNN based on FaceNet [11]. The API uses a pre-trained model called OpenFace to generate the 128 dimension vector [12].

Finally, the vector data is compared to data of a known face using an SVM classifier pre-trained by dlib. The result of the classifier will either be a match or an unknown face. If there is a match then a relay could be energized to allow a vehicle to start. Without a match the car would not start thus preventing it from being stolen. If a match is not found after a predetermined time then a known user can enter a pin code to use the vehicle.

Face Position Tracking will use the landmarks assigned from the HOG classifier to track a face based on where they appear in the video frame. Since the goal is to track a drivers face, a camera would be placed directly in front of the driver. Moving the head side-to-side is often necessary when driving to check your surroundings and the major distractions that cause accidents are texting or falling asleep. In both cases, a drivers head will droop downward so that is what will be tracked. When the head droops an alarm will sound notifying the driver. Additionally if a user's face is not located when tracking is active an alarm will sound. This is a case where a driver may be reaching for something and their head is not facing the road.

The overall system will execute in the following sequence:

1. Face Recognition executes on startup.
    a. If a match is found then the Face Tracking stage is entered.
    b. If a match is not found the user is prompted to enter a pin code. If the pin code is correct then Face Tracking begins, otherwise the program is terminated.
2. Face Tracking initially begins in the OFF state because a car is not in gear when started.
3. When the car is placed in gear and moving forward (simulated by a push button) then Face Tracking will be running.

This project also utilizes OpenCV software which is an Open Source Computer Vision Library. OpenCV is used to interface with the webcam as well as display the webcam feed allowing the user to visualize the results. The library has many built in functions which are very simple to implement. Additionally, the Jetson Nano GPIO library is used to communicate with the push button and speaker. There are built in functions set pins as input or output and to trigger a threaded callback function if an event is recognized; similar to an interrupt. This library is specific to the Jetson Nano board and would likely be omitted in a vehicle's version.

There is an additional program included to allow the user to take a picture with the webcam and upload it as a known face. This is done to test multiple users in the program. In a vehicle this would need to be implemented with additional security measures if the vehicle's camera were to be used. Another option would be to allow a registered user to upload an image using a smartphone or computer; assuming the vehicle has an App interface.

4. Block Diagrams

Face Recognition diagram:

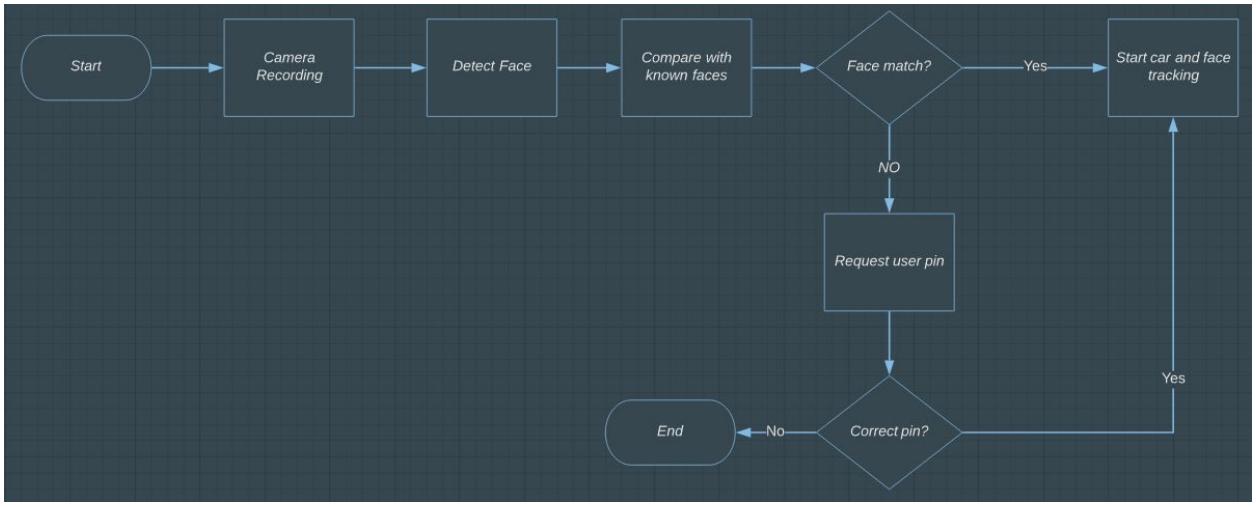


Figure 6: Facial Recognition System

Face Position Detection diagram:

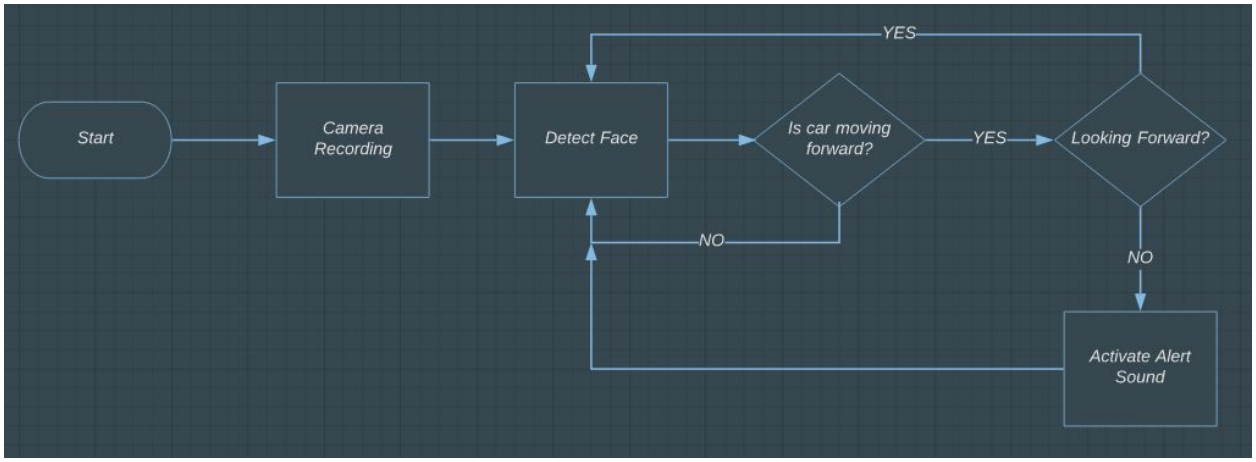


Figure 7: Face Position Detection System

5. Results

A simple Python script for taking a picture using the webcam is implemented using OpenCV library. The user is prompted to enter a name for the image and the picture is taken when the 'S' key is pressed on a keyboard. In a vehicle, this same process could be implemented using the console's touch screen.

The Face Recognition script reads a saved image and encodes it into a 128 dimension byte vector. The encodings are stored in a list which is used to compare with faces in the real-time video feed. **Figure 8** shows the encoding of each team member.

```
brian_image = face_recognition.load_image_file("bc.jpg")
brian_face_encoding = face_recognition.face_encodings(brian_image)[0]

jason_image = face_recognition.load_image_file("jason.jpg")
jason_face_encoding = face_recognition.face_encodings(jason_image)[0]

known_face_encodings = [
    brian_face_encoding,
    jason_face_encoding
]
known_face_names = [
    "Brian",
    "Jason"
]
```

Figure 8: Known Face encodings

The video feed is first resized to ¼ the size which leads to faster processing and then converted to RGB format. Next, the location of the face is detected and assigned values for the top, right, left, and bottom. The location data is stored in a structured numpy array format for later processing. When a face location is found, it is then encoded the same way as the saved images are. This process is shown in **Figure 9**.

```
face_locations = face_recognition.face_locations(rgb_small_frame)
face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)
```

Figure 9: Real-time face encoding

If encoded faces are present in the webcam feed they are compared to each known face from the saved images. A list of true or false for each face comparison is returned and used to match with

the name of the person when displaying results. **Figure 10** shows how this is accomplished. The compare_faces function returns the true/false result of each encoding. The face_distance function returns a euclidean distance for each encoding to indicate how similar face encodings are. A numpy function np.argmin is used to return indices of minimum value of the face distances.

```
for face_encoding in face_encodings:
    matches = face_recognition.compare_faces(known_face_encodings, face_encoding)
    name = "Unknown"

    face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
    best_match_index = np.argmin(face_distances)

    if matches[best_match_index]:
        name = known_face_names[best_match_index]
        match_found = True
```

Figure 10: Determine if a face has a match

Figure 11 shows the results are accurate even if the user is wearing a hat or sunglasses. If a match is not found after a few seconds, then a function call is made to ask the user for a pin code. If the pin is entered correctly the program continues to face tracking, otherwise it is terminated.

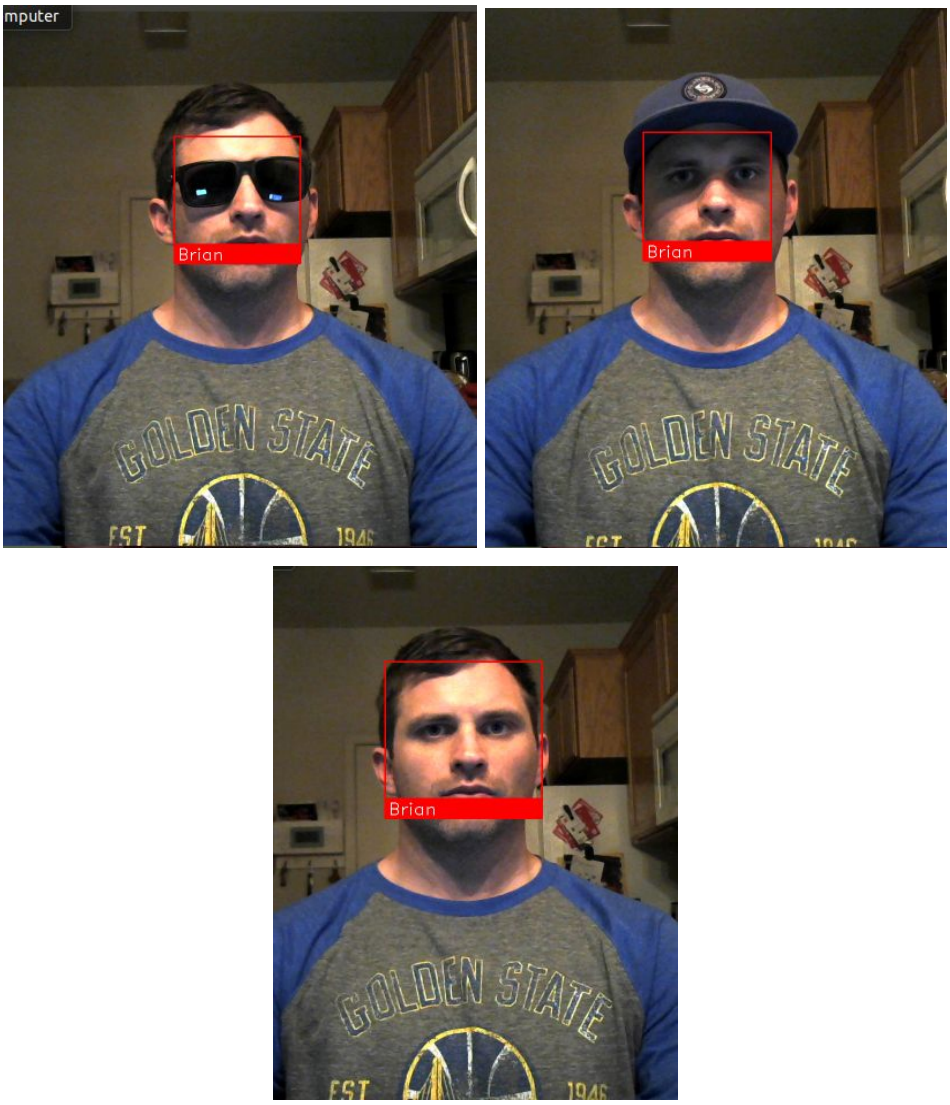


Figure 11: Face Recognition Results

Face Position Tracking is done using the same Face Recognition API library. However, there is no need to determine who the person is at this stage, therefore the encoding and comparing steps are omitted. In this case, the face location is determined in the same way as before and the face landmarks are stored in a list. The landmarks are represented as X and Y coordinates based on the face location data. Next, the minimum and maximum Y coordinate values for the nose and chin are determined using built in Python functions. The difference between the max and min is calculated and compared to a threshold value. The process is shown in **Figure 12**. The Y coordinate is using a lamba function to access the second index in a list of lists. Chin and nose bridge are lists of coordinates within the face landmarks list.

```
for face_landmarks in face_landmarks_list:

    chin = face_landmarks['chin']
    maxLandmark = max(chin, key = lambda i:i[1])[1]
    minLandmark = min(chin, key = lambda i:i[1])[1]
    chin_diff = (maxLandmark - minLandmark)/10

    nose_bridge = face_landmarks['nose_bridge']
    maxLandmark = max(nose_bridge, key = lambda i:i[1])[1]
    minLandmark = min(nose_bridge, key = lambda i:i[1])[1]
    nb_diff = (maxLandmark - minLandmark)/10

    if chin_diff> 4.9 and nb_diff > 1.7:
        cnt = cnt + 1
        if cnt > 8:
            status = "alert!!!"
            labelColor = RED
            GPIO.output(alert_pin, GPIO.HIGH)

    else:
        cnt = 0
        status = "driving"
        labelColor = BLUE
        GPIO.output(alert pin, GPIO.LOW)
```

Figure 12: Determine chin and nose orientation

The threshold was determined through experiment. The webcam was placed slightly beyond arm's length; which in a vehicle is roughly how far the driver is from the dash. The calculations of the Y coordinate values were monitored while moving the head in all directions. The threshold values are only exceeding when the head is facing down, as shown in **Figure 13**.

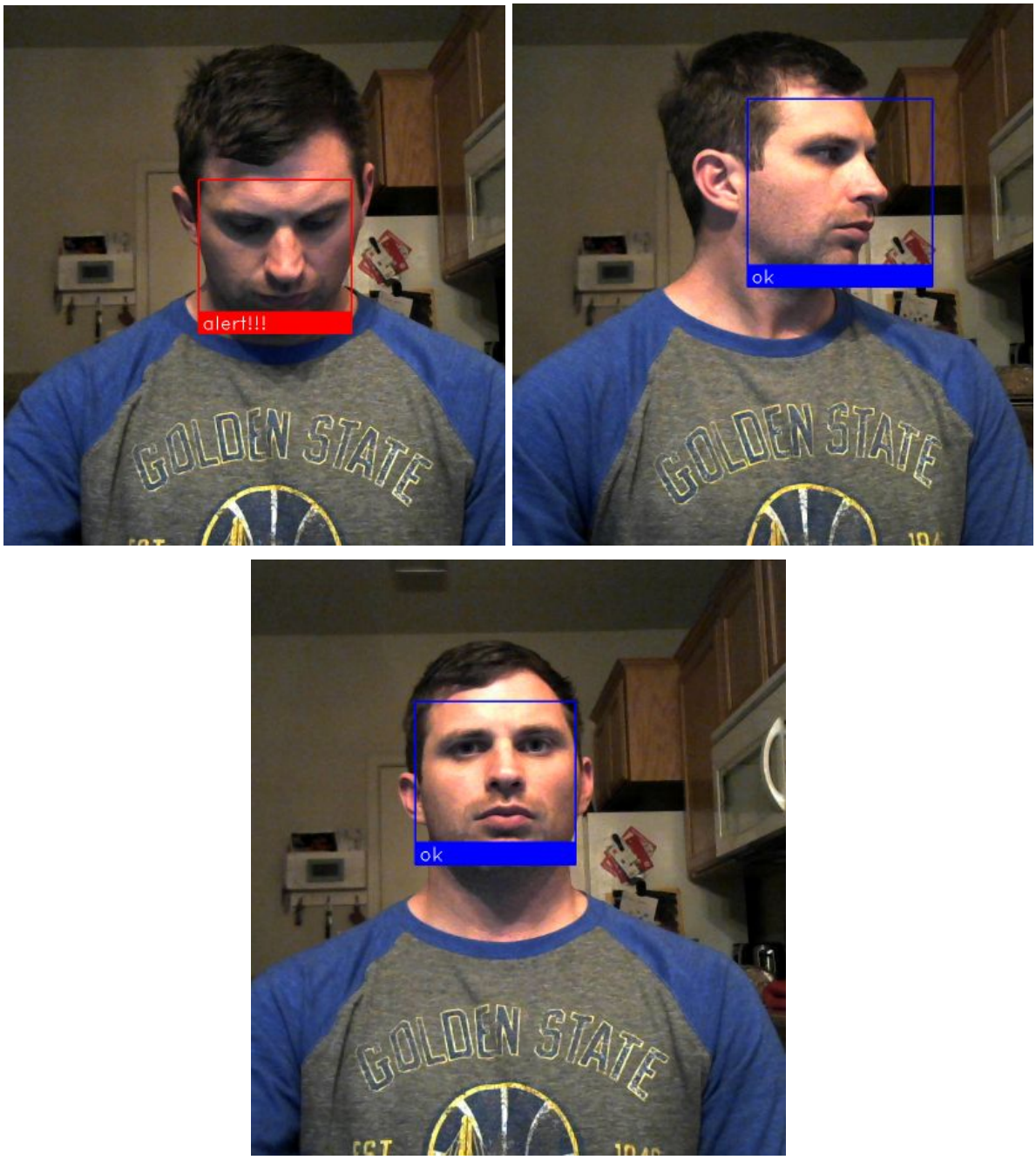


Figure 13: Position Tracking Results

This process is placed in an if/else decision statement and only executed if the drive state is true. The drive state is set based on a push button connected to the GPIO pins. **Figure 14** shows the GPIO pin functionality. After configuring the pins an event detect is implemented for the push button. The callback function driving_state simply inverts a boolean driving state variable. The

event detects callback function is executed in a separate thread which allows the tracking to continue in parallel.

```
# Pin Setup:
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)  # BOARD pin-numbering scheme
GPIO.setup(but_pin, GPIO.IN)  # button pin set as input
GPIO.setup(alert_pin, GPIO.OUT)
GPIO.add_event_detect(but_pin, GPIO.RISING, callback=driving_state, bouncetime=200)
```

Figure 14: GPIO pin functions

See Appendix A for complete code.

6. Parts and Budget

Required Parts	Cost
Jetson Nano	Free
Webcam	$20
Monitor	$55
SD Card	$10
HDMI Cable	Free

*Note: Items labeled as ‘free’ were owned by a member prior to the project.

7. Project Standards

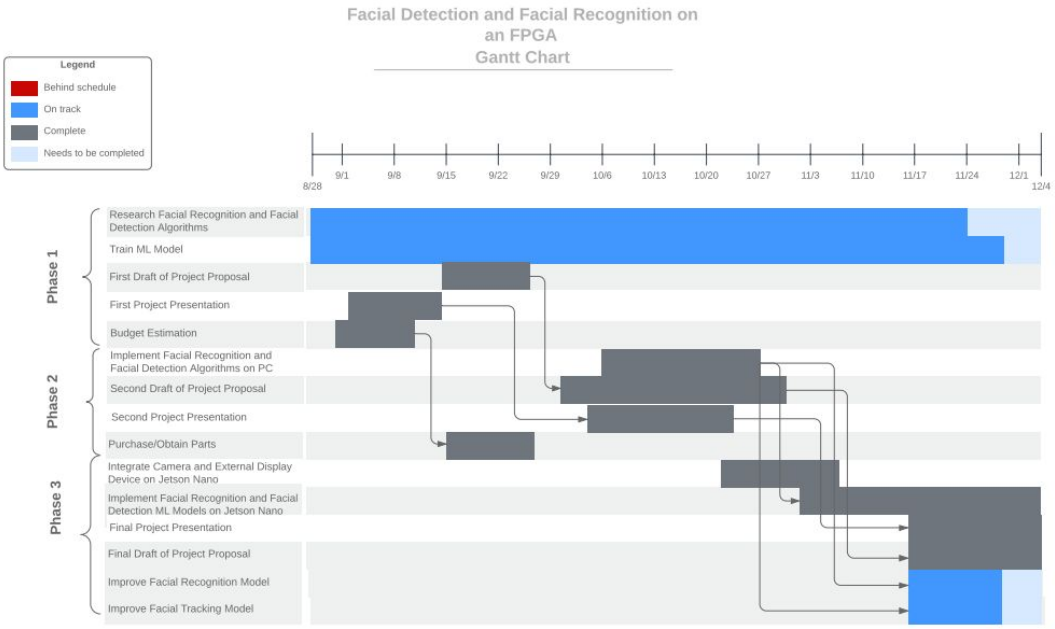
Artificial Intelligence and Machine Learning are relatively new fields and Industry Standards are an ongoing process. Currently there are no official IEEE standards for these disciplines. However, the IEEE P7000 series is a group of projects under development with the goal of standardizing Autonomous and Intelligent Systems[3].

Additionally, on February 11, 2019 the President of the United States issued Executive Order 13859 for Maintaining American Leadership in Artificial Intelligence[4]. The Executive Order directs the National Institute of Standards and Technology (NIST) to develop technical standards, safe testing, and deployment of AI technologies. On August 9, 2019, NIST released “A Plan for Federal Engagement in

Developing Technical Standards and Related Tools" in response to the Executive Order. It calls for the government to "speed the pace of reliable, robust, and trustworthy AI technology development."

Based on government organizations currently implementing standards and consumer demand of AI technologies the field is going to continue its rapid growth. It is such a powerful technology that the need for strict standards are extremely important to developing and maintaining safe and reliable AI systems.

8. Schedule



9. References

[1] National Highway Traffic Safety Administration. (April 2019) *Distracted Driving Fatal Crashes, 2017* [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812700.

[2] National Highway Traffic Safety Administration. (Oct 2017) *Drowsy Driving, 2015* [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812446.

[3] IEEE Standards Association. *The IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems* [Online]. Available: https://standards.ieee.org/industry-connections/ec/autonomous-systems.html.

[4] National Institute of Standards and Technology (NIST). (May 2019) *Artificial Intelligence Standards* [Online]. Available: https://www.federalregister.gov/documents/2019/05/01/2019-08818/artificial-intelligence-standards.

[5] Github face_recognition. (2017). [Online]. Available: https://github.com/ageitgey/face_recognition

[6] Labeled Faces in the Wild. University of Massachusetts. (Oct. 2019).[Online]. Available: http://vis-www.cs.umass.edu/lfw/results.html

[7] Deep Residual Learning for Image Recognition. (2016). [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

[8] Histograms of Oriented Gradients for Human Detection. (2005). [Online]. Available: http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf

[9] Face Point Annotations. [Online]. Available: https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

[10] One Millisecond Face Alignment with an Ensemble of Regression Trees. (2014). [Online]. Available: http://www.csc.kth.se/~vahidk/papers/KazemiCVPR14.pdf

[11] FaceNet: A Unified Embedding for Face Recognition and Clustering. (2015). [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2015/app/1A_089.pdf

[12] OpenFace. (2016). [Online]. Available: https://cmusatyalab.github.io/openface/models-and-accuracies/

[13] "Summed-Area Table." *Wikipedia*, Wikimedia Foundation, 21 Sept. 2019, Available: en.wikipedia.org/wiki/Summed-area_table.

[14] "Kanade–Lucas–Tomasi Feature Tracker." *Wikipedia*, Wikimedia Foundation, 5 Aug. 2019, Available: en.wikipedia.org/wiki/Kanade%E2%80%93Lucas%E2%80%93Tomasi_feature_tracker.

[15] “Facts Statistics: Auto theft,” III. [Online]. Available: https://www.iii.org/fact-statistic/facts-statistics-auto-theft. [Accessed: 16-Dec-2019].

11. Appendix

Appendix A: face_recognition_tracking.py

```
import face_recognition
import cv2
import numpy as np
import time
import Jetson.GPIO as GPIO

drive = False

def recognition():
    video_capture = cv2.VideoCapture(0)


    brian_image = face_recognition.load_image_file("bc.jpg")
    brian_face_encoding = face_recognition.face_encodings(brian_image)[0]

    jason_image = face_recognition.load_image_file("jason.jpg")
    jason_face_encoding = face_recognition.face_encodings(jason_image)[0]

    known_face_encodings = [
        brian_face_encoding,
        jason_face_encoding
    ]
    known_face_names = [
        "Brian",
        "Jason"
    ]

    face_locations = []
    face_encodings = []
    face_names = []
    match_found = False
    start_time = time.process_time()
    while (time.process_time() - start_time) < 5.0:
        ret, frame = video_capture.read()

        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
```

```
        rgb_small_frame = small_frame[:, :, ::-1]

        face_locations = face_recognition.face_locations(rgb_small_frame,
model="cnn")
        face_encodings = face_recognition.face_encodings(rgb_small_frame,
face_locations,model="large")


        face_names = []
        for face_encoding in face_encodings:
            matches = face_recognition.compare_faces(known_face_encodings,
face_encoding,tolerance=0.6)
            name = "Unknown"

            face_distances =
face_recognition.face_distance(known_face_encodings, face_encoding)
            best_match_index = np.argmin(face_distances)

            if matches[best_match_index]:
                name = known_face_names[best_match_index]
                match_found = True

            face_names.append(name)

        # display results
        for (top, right, bottom, left), name in zip(face_locations,
face_names):
            top *= 4
            right *= 4
            bottom *= 4
            left *= 4

            cv2.rectangle(frame, (left, top), (right, bottom), (0, 0,
255), 2)

            cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0,
0, 255), cv2.FILLED)
            font = cv2.FONT_HERSHEY_DUPLEX
            cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0,
(255, 255, 255), 1)

        sframe = cv2.resize(frame,(640,480))
        cv2.imshow('Face Recognition running...', sframe)
        cv2.moveWindow('Face Recognition running...',0,0)
```

```
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    if not match_found:
        print("match not found.")
    else:
        print("match found.")

    video_capture.release()
    cv2.destroyAllWindows()
    return match_found


def wait_to_drive():
    global drive


    but_pin = 18  # Board pin 18
    alert_pin = 16  # Board pin 16

    # Pin Setup:
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BOARD)  # BOARD pin-numbering scheme
    GPIO.setup(but_pin, GPIO.IN)  # button pin set as input
    GPIO.setup(alert_pin, GPIO.OUT)
    GPIO.add_event_detect(but_pin, GPIO.RISING, callback=driving_state,
bouncetime=200)


def driving_state(channel):
    global drive
    drive = not drive


def tracking():
    global drive

    video_capture = cv2.VideoCapture(0)
    RED = (0, 0, 255)
    GREEN = (0, 255, 0)
    BLUE = (255, 0, 0)
    alert_pin = 16  # Board pin 16
    face_locations = []
    status = ''
    labelColor = BLUE
```

```
    cnt = 0

    while True:
        ret, frame = video_capture.read()

        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

        rgb_small_frame = small_frame[:, :, ::-1]

        face_locations = face_recognition.face_locations(rgb_small_frame)
        face_landmarks_list =
face_recognition.face_landmarks(rgb_small_frame, face_locations)

        if drive:
            if not face_landmarks_list:
               GPIO.output(alert_pin, GPIO.HIGH)
                status = "alert!!!"
                cv2.rectangle(frame, (100,60),(1,1),RED,-1)
                cv2.putText(frame, status, (5,30),
cv2.FONT_HERSHEY_DUPLEX, 1.0, (255, 255, 255), 1)
            for face_landmarks in face_landmarks_list:

                chin = face_landmarks['chin']
                maxLandmark = max(chin, key = lambda i:i[1])[1]
                minLandmark = min(chin, key = lambda i:i[1])[1]
                chin_diff = (maxLandmark - minLandmark)/10

                nose_bridge = face_landmarks['nose_bridge']
                maxLandmark = max(nose_bridge, key = lambda i:i[1])[1]
                minLandmark = min(nose_bridge, key = lambda i:i[1])[1]
                nb_diff = (maxLandmark - minLandmark)/10

                if chin_diff> 4.9 and nb_diff > 1.7:
                    cnt = cnt + 1
                    if cnt > 8:
                        status = "alert!!!"
                        labelColor = RED
                        GPIO.output(alert_pin, GPIO.HIGH)

                else:
                    cnt = 0
                    status = "driving"
                    labelColor = BLUE
                    GPIO.output(alert_pin, GPIO.LOW)
```

```
        else:
            status = "NOT driving"
            GPIO.output(alert_pin, GPIO.LOW)

        for (top, right, bottom, left) in face_locations:
            top *= 4
            right *= 4
            bottom *= 4
            left *= 4


            cv2.rectangle(frame, (left, top), (right, bottom), labelColor,
2)
            cv2.rectangle(frame, (left, bottom - 35), (right, bottom),
labelColor, cv2.FILLED)
            cv2.putText(frame, status, (left + 6, bottom - 6),
cv2.FONT_HERSHEY_DUPLEX, 1.0, (255, 255, 255), 1)



        sframe = cv2.resize(frame,(640,480))
        cv2.imshow('Face Tracking running...', sframe)
        cv2.moveWindow('Face Tracking running...',0,0)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    video_capture.release()
    cv2.destroyAllWindows()
    print("stop")

def pin_code():
    pin = 1234
    codeEntered = input("Enter pin code: ")
    if codeEntered == str(pin):
        print("Success")
        return True
    else:
        print("Incorrect pin")
        return False

def main():
    match = recognition()
    if not match:
        for i in range(5):
```

```
            match = pin_code()
            if match:
                break
    if match:
        wait_to_drive()
        tracking()
        GPIO.cleanup()

    print("System turned off")

if __name__ == '__main__':
    main()
```

Appendix B: take_picture.py

```
import cv2

fname = input("Enter file name: ")
fname = fname + '.jpg'
webcam = cv2.VideoCapture(0)
while True:
    try:
        ret, frame = webcam.read()
        sframe = cv2.resize(frame,(640,480))
        cv2.imshow('Picture', sframe)
        cv2.moveWindow('Picture',0,0)
        if cv2.waitKey(1) & 0xFF == ord('s'):
            cv2.imwrite(filename=fname, img=frame)
            webcam.release()
            cv2.destroyAllWindows()
            break
        elif cv2.waitKey(1) & 0xFF == ord('q'):
            webcam.release()
            cv2.destroyAllWindows()
            break

    except(KeyboardInterrupt):
        webcam.release()
        cv2.destroyAllWindows()
        break
```