DreamCatcher

George Serge Barsukov (gsb@ischool.berkeley.edu)
Murat Melek (murat.melek@berkeley.edu)
Sai Ravuru (sairavuru@berkeley.edu)

Introduction

Project DreamCatcher is an application for autonomous vehicles that prevents drivers from falling asleep while driving on **"no-sleep**" zones of the highway system.

Motivation

Companies such as UBER and TESLA are working on autonomous vehicles. As of the day of the completion of this white paper, level of autonomy for personal automobiles is limited to automatic emergency braking and lane control (Kaslikowski, 2019). Significant social and technological advances need to be done before fully computer controlled personal automobiles with no human intervention are on the roads.

One major issue that is a point for discussion is the level of autonomy. Will the driver have complete freedom to do whatever he/she wants? Should the driver be able to sleep during the journey? An intermediate step could be designating full-autonomy zones where computer takes complete control of the vehicle.

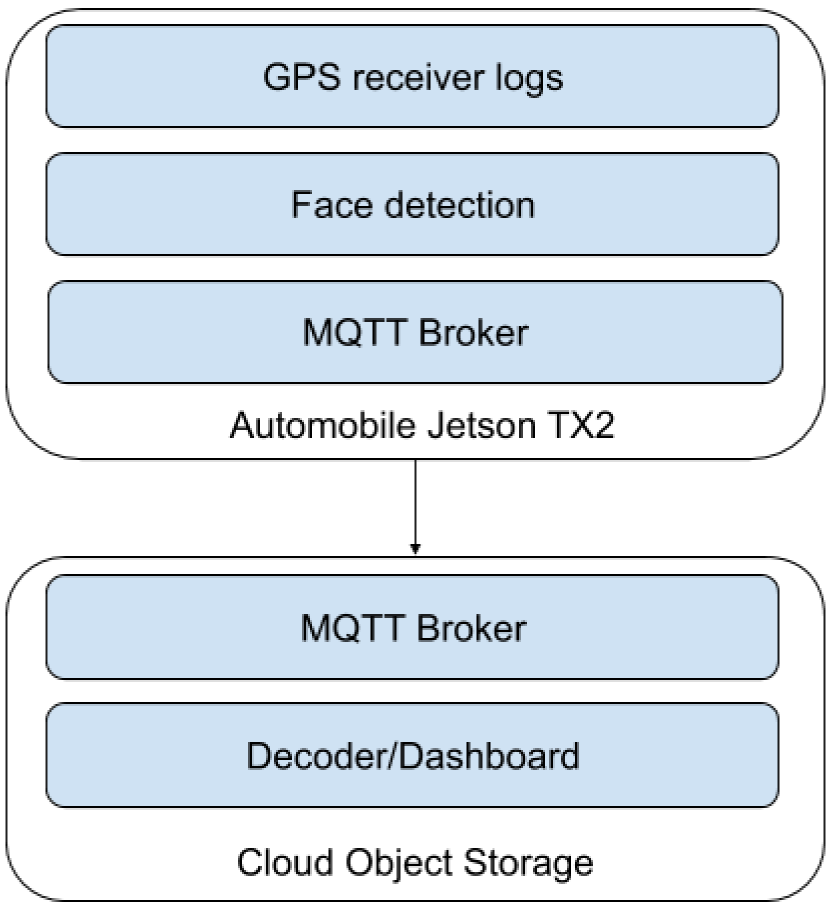
German Federal-Controlled Highway System (Autobahn) could be used as an example for full-autonomy zones. Certain segments of the Autobahn are designated as "speed-unrestricted" zones. No speed limit is applied to these segments for some classes of vehicles. Similar to the Autobahn, could there be certain segments of the freeway system that are "**autonomy-unrestricted**"?

For this type of an approach, we would need a system that:

I. Keeps an eye on the driver tracking the driver's concentration to the road.
II. If the driver is unable to concentrate (e.g. sleeping), make sure that the vehicle is travelling on a autonomy-unrestricted segment.

System architecture

The infrastructure, as shown below, is broken down into two components. The first component is designed to subscribe to GPS receiver logs. This component lives on the edge device and monitors the user to detect whether or not he/she is sleeping. If a sleeping user is detected, it pushes an alert message onto its MQTT broker. On the second component exists another broker which subscribes to the edge broker. Once messages are received, they are decoded into a picture with metadata. The decoded message is then uploaded into cloud storage in an IBM cloud storage bucket. Furthermore, the image is loaded into an html dashboard on which authorities can identify the violators.



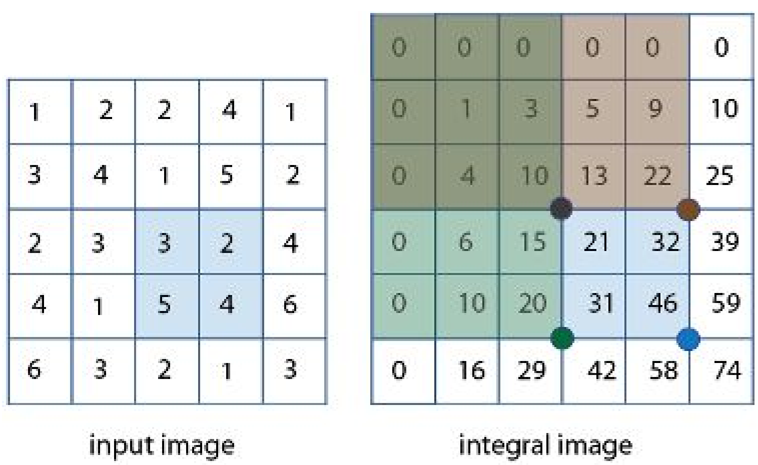
Face detection

In the core of our application, we have app.py. This is an edge application that will be running on our Jetson TX2. This program handles a live camera feed and uses it to detect if there is a sleeping driver within the stream. The program begins by reading in the driver's GPS location, as well as his/her speed. The program then periodically checks the video feed to detect faces using a HAAR Cascade Classifier (more detail on this later). If a face is detected, the application will then run another HAAR Cascade Classifier: Haarcascade_eye_tree_eyeglasses (Hameed, 2013) to detect whether the user's eyes are closed.

As soon as this event is detected, the system will form an alert message. This message consists of the image of the sleeping driver, along with some metadata including the speed, and datetime. The message is saved to disk to be further processed as well as propagated into a remote broker where it will be logged.

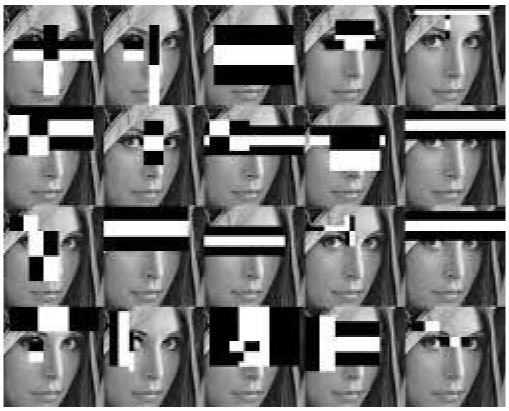
Since the HAAR Cascade Classifier algorithm is a large part of the application, it's important to understand how it works. It's good at identifying a specific object within an image and doing so quickly. In our case, we used a pre-trained version of a face detection model and an eye detection model. Regardless, we'll go through the details of the model creation.

The first step for constructing them is to gather a set of positive and negative examples of the object being classified. In our case, these images would have been of eyes and faces. Next, we compute integral images out of each of these examples. An integral image merely sums all pixels above and to the left of the selected pixel and replaces it's value with that sum. The purpose of this step is to reduce the amount of computation required in a future step.



From the image above, we can compute the sum of the selected box with very few calculations, the sum would be 46-20-22+10=14 (we can confirm this with 3+2+5+4=14). The values come from subtracting the area of both the rectangle to the left, which is already computed in its bottom right corner, as well as the rectangle on the top. We then need to add back the area of the square on the top left because we had subtracted it out twice. While this example doesn't show improved performance since we did 4 computations either way, when we increase the sizes of the shapes we'll greatly reduce the number of computations necessary.

After we've converted our images, we can begin to compute simple features. These features are as simple as taking some area in the image and subtracting it from another area. They're very simple yet very numerous and the integral images from the last step lets us compute them efficiently. Once we have our large set of features, we use **Adaboost** algorithm to combine our *weak* features into *strong* features. That is to say, features that maximally separated the positive examples from the negative. However, these features are not strong enough to classify the image on their own, so they're used like a cascade of filters where in order for an object to be identified in the image, it must pass all filters. The steps are all independently simple and the model's compute time is quick, yet it still performs well.

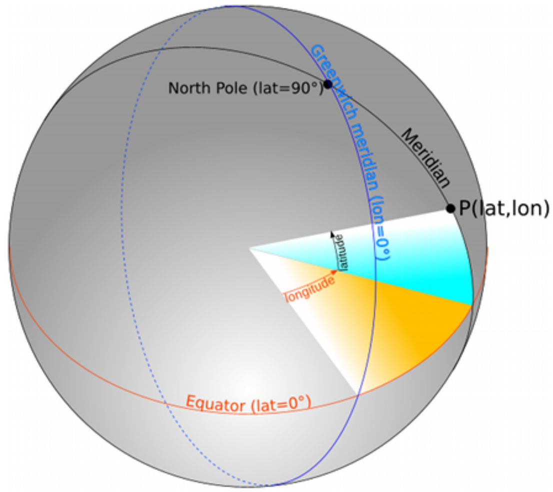


When we want to detect multiple objects, we can just use two different models over the same image to detect them. In our case, we run them serially since we have no use in running the eye detection portion without first detecting that there's a face.

Overall, this algorithm fits our use case because we're processing real time streams and we need quick performance. We need to identify the driver's face and eyes and determine whether or not he/she is sleeping.

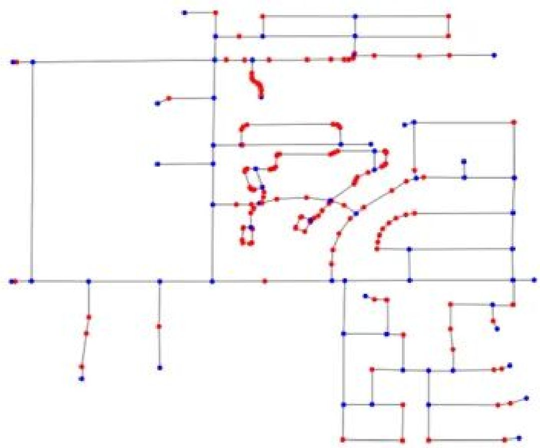
GPS and Traffic computation

In order to determine if a speed or sleep zone is encroached, the GPS position along with the corresponding speed is required. In addition, sleep and speed zones are required to be overlaid on a map around the current position. Using the GPS coordinates [latitude, longitude] and the datetime stamp from the GPS receiver is relayed to logs.



A geocoding open-source library called Geopy is used to read the GPS coordinates from the logs to compute the distance between them. Together with the time of travel between 2 successive points over chronological datetime stamps in the logs, the speed is computed.

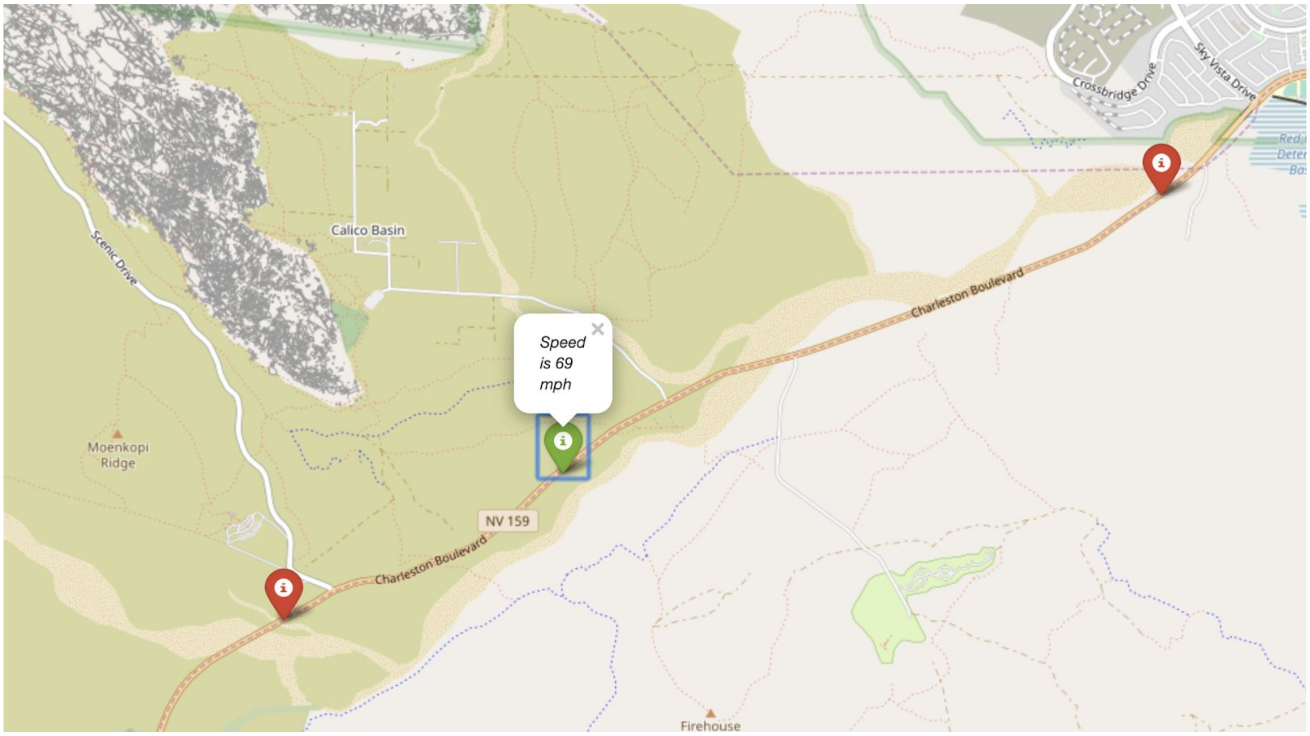
For each GPS coordinate, an open-source mapping library called osmnx(OpenStreetMap networkx) is used to extract a surface level grid of the surrounding area [10 mile radius]. This gridded area as seen below is converted to nodes and edges in order to further filter for roads and highways with speed limits ["maxspeed"].



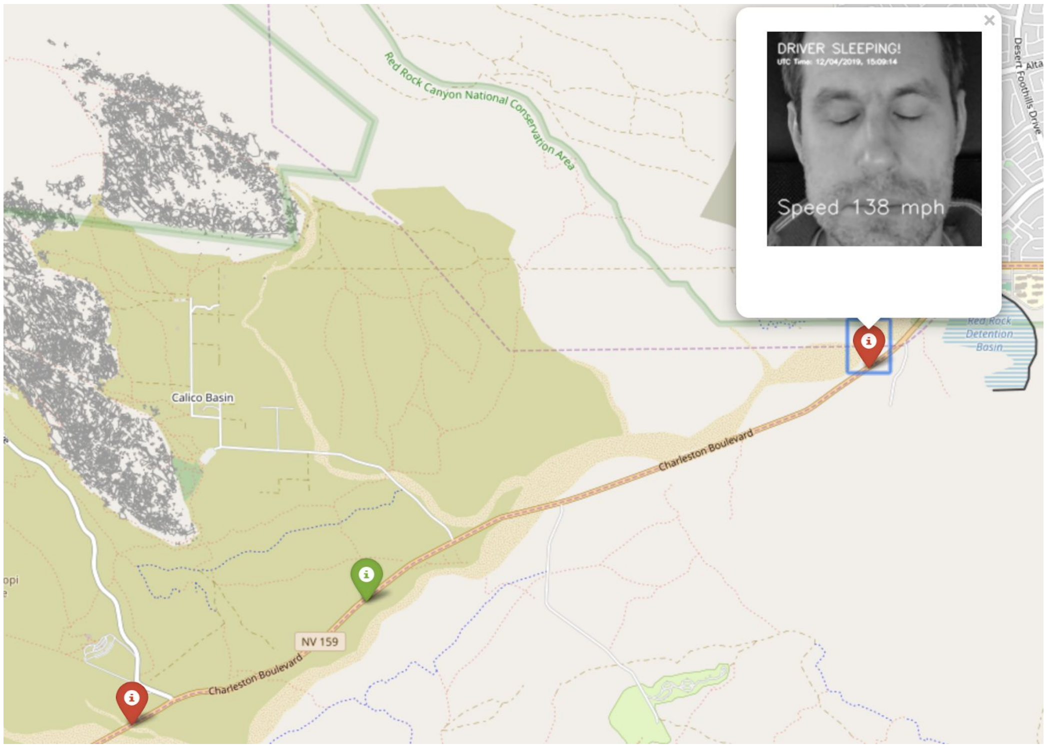
A sample list of features are as shown in the following code snapshot where geopandas dataframes are the result of the networkx nodes & edges when converted from the grid data.

```
In [12]: nodes, edges = ox.graph_to_gdfs(graph)

In [13]: nodes.head()
Out[13]:
           highway     osmid   ref        x        y  \
25216594       NaN  25216594   NaN   24.921  60.1648
25238874       NaN  25238874   NaN   24.921  60.1637
25238883  crossing  25238883   NaN  24.9214  60.1634
25238933  bus_stop  25238933  1168  24.9245  60.1611
25238944       NaN  25238944   NaN  24.9213  60.1646

                             geometry
25216594  POINT (24.9209884 60.1647959)
25238874  POINT (24.9210331 60.1636625)
25238883  POINT (24.9214283 60.1634425)
25238933   POINT (24.924529 60.1611136)
25238944   POINT (24.921303 60.1646301)

In [14]: edges.head()
Out[14]:
  access bridge                                          geometry   highway  \
0    NaN    NaN  LINESTRING (24.9209884 60.1647959, 24.9208687 ...   primary
1    NaN    NaN  LINESTRING (24.9209884 60.1647959, 24.9209472 ...   primary
2    NaN    NaN  LINESTRING (24.9210331 60.1636625, 24.9210408 ...   primary
3    NaN    NaN  LINESTRING (24.9214283 60.1634425, 24.9214018 ...   primary
4    NaN    NaN  LINESTRING (24.9214283 60.1634425, 24.9210916 ...  cycleway

   key lanes     length maxspeed           name  oneway  \
0    0     2   6.654290       40   Porkkalankatu    True
1    0     2  40.546678       40  Mechelininkatu    True
2    0     2   5.597971       40  Mechelininkatu    True
3    0     4  16.322546       40  Mechelininkatu    True
4    0   NaN  18.647504      NaN             NaN   False

                osmid service tunnel         u           v
0            23717777     NaN    NaN  25216594  1372425721
1  [23856784, 31503767]     NaN    NaN  25216594  1372425714
2            29977177     NaN    NaN  25238874   336192701
3            58077048     NaN    NaN  25238883   568147264
4           160174209     NaN    NaN  25238883   258190363

In [15]: type(edges)
Out[15]: geopandas.geodataframe.GeoDataFrame
```

If the current speed of the vehicle exceeds the maximum speed limit of the surround area, the speed criteria is breached. In this case, the speed is visualized on the map tracking the movements of the vehicle as seen below. If the speed is below the speed limit, the icon is green. If the current speed is above the speed limit, the icon is red.



If the speed of the area is below 40mph, the area is considered to be in a no sleep zone. If the speed is breached in this zone, the sleep criteria is breached as well. In this case, the speed and datetime are stamped onto the sleeping snapshot from the classifier. In addition, the picture is added to the visual dashboard on the GPS coordinates along the vehicle line of travel as shown below. The visual dashboard acts as a tool for authorities to assist them in locating violators and enforcing traffic laws.



Future feature additions

Additional features can be added to the scope of this infrastructure and requirements such as:

a. GPS receiver live feed
b. Connect system to sleep detection to vehicle warning systems
c. Enable city and highway planners to determine city edges
d. Live traffic conditions can be de-centralized and fed to traffic optimization models
e. Autopilot cars can be auto-configured for long-drive(no sleep or speed zones) comfort vs. short-drive(in sleep or speed zones) attention modes

The above-mentioned feature additions however requires the buy-in of both authorities and car manufacturers which is possible in the near-future.