# Machine Learning Based Action Recognition to Understand Distracted Driving

by

Andrew Radlbeck

A Thesis
Submitted to the Faculty
of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Master of Science
in
Electrical and Computer Engineering
by

December 2019

APPROVED:

Professor Alexander Wyglinski, Research Advisor

Professor Donald Richard Brown, Committee Member

Professor Xinming Huang, Committee Member

## Abstract

The ability to look outward from your vehicle and assess dangerous peer behavior is typically a trivial task for humans, but not always. Distracted driving is an issue that has been seen on our roadways ever since cars have been invented, but even more so after the wide spread use of cell phones. This thesis introduces a new system for monitoring the surrounding vehicles with outside facing cameras that detect in real time if the vehicle being followed is engaging in distracted behavior. This system uses techniques from image processing, signal processing, and machine learning. Its ability to pick out drivers with dangerous behavior is shown to be accurate with a hit count of 87.5%, and with few false positives. It aims to help make either the human driver or the machine driver more aware and assist with better decision making.

# Acknowledgements

The path to perform this research and to bring the system to function in the real world could not have been done by myself alone. I would like to thank my adviser, Dr. Alexander Wyglinski, and my committee professors Dr. Donald Richard Brown and Dr. Xinming Huang, for their support and guidance during this time. This thesis has taken many turns and it has been a tremendous learning experience for me and hopefully everyone who takes up the journey to read it. I would like to thank my wife, Evita Vigante, for her support in this effort and my daughter Silvia for motivating me to get things done in a timely manner. It would also not be possible without my stunt driving team comprising of my wife, brother and law, Arthur Vitkovskis, my adviser, and the countless others I have filmed along the way.

# Contents

# List of Figures

# List of Tables

# List of Acronyms

**AI** Artificial Intelligence

**CNN** Convolutional Neural Network

**CPU** Central Processing Unit

**GAN** Generative Adversarial Networks

**GPU** Graphics Processing Unit

**LSTM** Long Short Term Memory

**FFT** Fast Fourier Transform

**ML** Machine Learning

**NN** Neural Network

**PDF** Probability Density Function

**ReLU** Rectified Linear Unit

**RL** Reinforced Learning

**RV** Random Variable

**SGD** Stochastic Gradient Descent

**SVM** Support Vector Machine

**t-SNE** t-distributed Stochastic Neighbor Embedding

**PCA** Principal Component Analysis

**ROC** Receiver Operating Characteristic

**TP** True Positive

**TN** True Negative

**FP** False Positive

**FN** False Negative

# Chapter 1

# Introduction

## 1.1 Motivation

Distracted driving is to blame for 3,166 deaths in 2017, accounting for 9% of all crashes [1]. A large part of the increase is due to how users interact with their phones. Cell phone use has moved from talking on the phone and looking at the road to looking at the phone and interacting with the screen [2]. Distraction related crashes have always been a problem, but the reason for these crashes has been increasingly caused by cell phones [1, 3–5].

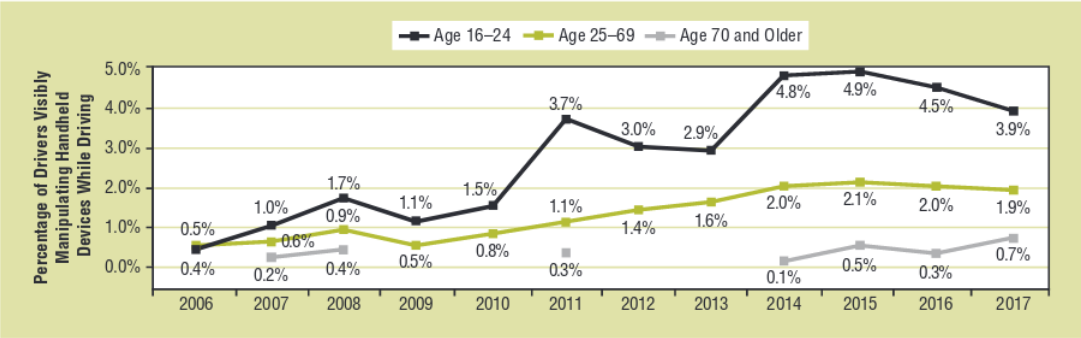



Figure 1.1: The information in this figure is obtained from [2]. Note that there was insufficient data to fill in all points for each age group. Cell phones continue to be the culprit behind distracted driving, and it is becoming increasingly responsible for distraction related crashes.

The problem is exacerbated among teen drivers and is the cause of more than 58% of crashes involving them [6]. These accidents cause harm not just to the distracted driver, but to others as well. Cell phones are not the only cause for distracted behavior and we look to address all type of distraction in this work. Technology that detects distracted driving has potential to reach 272 million drivers that are on the road today in the United States [7].

Traffic flow studies have shown that distracted drivers exhibit an increase in certain road behaviors that include lane deviations and speed fluctuations. They will also exhibit a decrease in lane changes [8]. These behaviors create dangerous road conditions, inefficient traffic flow patterns, unease among road users, and raises the likelihood for traffic and slowdowns.

The primary motivations of this thesis are to add new abilities to cars existing today, legacy vehicles, and to bring this technology to self-driving cars in order to help make better planning decisions, as well as impact the users through increased safety and traffic throughput. Finally, this work will help create awareness for this issue by highlighting the dangers and advantages; hoping to eliminate this hazardous social norm.

## 1.2 Similar and Influential Work

Most distracted driver detection techniques use an inward facing camera [9] and internal sensors [10] to determine if the person driving is paying attention. The goal of these techniques is mostly to complement internal driver assistance tools, such as distracted vehicle takeover [11], drowse vehicle takeover [12] and distracted lane keeping [13].

The goal of this work is a combination of research that exists today. However, we aim to solve the problem by looking outward at peer vehicles. The following contributions have laid the groundwork, and each give a different perspective on solving the problem of distracted driving. Specifically, the following work categorizes the type of distracted behavior, behavior prediction, creates a driver behavior model framework, covers learning algorithms for driver style, patterns, and ability, and finally covers types of inputs that have been used to make these classifications.

Vaitkus [14] categorized the behavior of the driver as normal or aggressive using only

accelerometer data. This paper is operating along a similar line of thought as the proposed solution, albeit, viewing the driver in the vehicle equipped with sensors and not its peers. The method used signal feature extraction to look at total of 39 features from each axis of a three-axis accelerometer. The features range from the minimum and maximum value of the signal to Kendalls tau rank correlation coefficient (for the full list see paper). These features were used with $k$-nearest neighbors classification algorithm and they were able to achieve a classification accuracy of 100%. Kuge [15] also did something similar with steering angle data, but instead trained a Hidden Markov Model and used data exclusively from a simulation environment. This hidden Markov Model was trained to recognize emergency lane changing, normal lane changing and lane keeping.

Fridman [9] approaches the problem with image processing and deep learning. The paper focuses on trying to detect where the driver is looking with an on-board camera viewing the driver. The method takes the video input, performs face detection, extracts features, and uses a random forest classifier to categorize their attention into several categories in order to determine if the driver's attention is on the road or not. The system created is capable of achieving 91.4% accuracy operating at 11 Hz. Companies have already begun using similar technology to manage business operated vehicles and monitor employees while they drive [16].

The data set introduced by the Honda Research institute [17] labels not just the actions the driver took, but also the reasons and gave contextual understanding behind those actions of the driver. For example, labels exist that describe what the other vehicles and pedestrians are doing in the scene. The data was then used to train a CNN (convolutional neural network), using image date, and an LSTM network (long short-term memory), using sensor data, to understand the context of the situation along with the driver actions. This context helped to give reasoning for predicting the driver's future behavior. Deo [18] also used an LSTM network except for trajectory estimation. This is somewhat like behavior prediction, except you are getting straight to the root of what you really need. The vehicles position in the future. They used a simulation environment with pre-recorded data and were able to obtain perfect location information from the surrounding vehicles. The network was able to achieve its goal with a low error.

Martinezs [10] survey paper is a great reference to see what methods where used on what driving related tasks. It helps to break down definitions for driving style, pattern and ability, as well as provide guidance on a driver behavior model framework. It explicitly lists all sensors commonly used and all driving style recognition class types that have been defined across a variety of other papers. It then proceeds to present results on the use of rule or model-based classification as well and supervised and unsupervised learning methods. The analysis shows that rule and model-based methods are the least effective and it is sometimes difficult to justify their accuracy. Learning based methods prove to work the best with little difference between supervised and unsupervised.

Ricci [19] has a patent which assesses the performance and ability of the driver and rates them. This is done with a suite of onboard sensors which attempts to figure out if laws have been broken, if the vehicle was stolen and if the driver if following proper road etiquette. These features are then used to score the driver and store this score in a database, where the driver will be given a reputation based on how well they drive. The aim here is to add accountability for the driver's actions and to keep a history of the types of choices they make.

## 1.3 Current Issues

Much of the technical difficulty in the field such as detection [20] and tracking [21] has been solved in a well-tested and accurate manner. In addition, many different methods using signal processing and machine learning on their works collected features have been explored. The difficulty for this work comes in three main places.

First, to have a real time system able to detect and track in easily accessible hardware. This will allow the system to be easily deployed on legacy vehicles. Second, the information gained through image processing has not yet been used to attempt distraction classification of peer vehicles. As shown above, the information for doing so is typically on the vehicle itself. Third, gathering an adequate dataset is always a difficult task. As mentioned in many places [22], the difficulty in any driving related task is that the interesting edge cases happen infrequently.

## 1.4 Thesis Contributions

This work will look to solve some of the issues above and add the following contributions:

1) A hardware test bed with adequate on-board processing to take in video, detect and track vehicles, and preform general processing. An inexpensive mix of COTS components helps to make this research accessible and portable allowing for easy data collection and in car mounting.

2) An image and signal processing expert solution for detecting distracted driving artifacts. Data collection for machine learning is difficult. This work allows us two advantages. First, it assists in training by providing an accurate, but not fully trustable, source of automated labeling. Second, it allows us to easily go back to our collected data and find events of interest which are used to both tune and train.

3) An online machine learning framework to learn from and complement the expert system. The proposed work looks to take advantage of machine learnings ability to understand subtleties in data and increase the overall effectiveness of the system.

## 1.5 Thesis Organization

The goals of each chapter are as follow:

Chapter 2 will break down the different processing methods used in the analysis of the video which are used to extract features. The next sections will look at the signal processing and machine learning methods used to turn those features into classifications.

Chapter 3 will go over the hardware system and how the different processing tasks are split up among the acceleration components. It will also address the difficulties of running the system in real time.

Chapter 4 will give an overview of the entire processing pipeline and explain the decision process in full detail. Next, it will talk about the types of driving scenarios the system hopes to detect along with some real-world examples. Finally, results will be presented and discussed.

Chapter 5 will conclude the work, its effectiveness and needed improvements and extensions.

# Chapter 2

# Processing Methods

The processing methods described below are used to create a pipeline from video input to an annotated video output. These annotations highlight the vehicles considered to be distracted to aid the driver or self-driving system. The methods where chosen to function within performance criteria as described in Chapter 3. A block diagram of the pipeline is shown below, and its components will be described in the following sections.

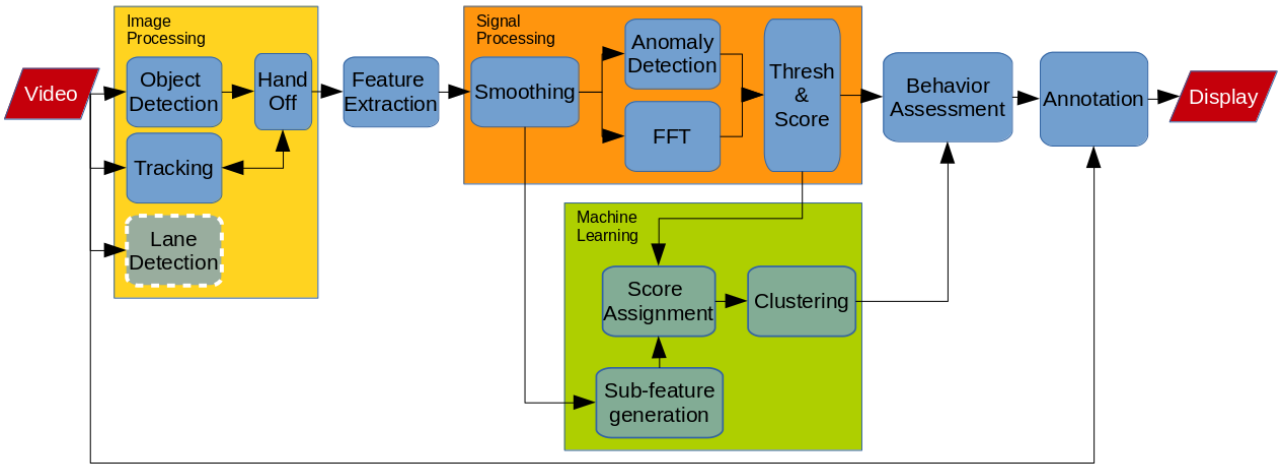



Figure 2.1: The flow of information moves left to right in the block diagram. Starting with the camera and ending with the display. The code architecture will be described later, but this will be a good guide for this chapter.

## 2.1 Image Processing

### 2.1.1 Detection

The last decade has been full of advancements in image processing, the most notable of which was with the use of the convolutional neural network (CNN) to solve problems with object recognition [23]. This deep learning revolution in the image processing space had kick-started the high-performance detection algorithm that made this work more accessible.

Detection, not to be confused with image classification, is the act of finding a certain object within an image. To perform detection, both image classification and object discovery need to take place.

The MobileNet V2 SSD [24] was created to have the highest accuracy possible while running on embedded processors such as the ones used on mobile phones. The MobileNet has opened the door for object detection to be deployed on many existing devices and at a low cost. Let us break down what the MobileNet V2 SSD is. MobileNet [25] is a CNN based neural network created specifically for mobile and embedded computer vision tasks. The main contribution here is the use of depthwise separable convolutions to replace the standard convolution layer. This replacement is made to reduce the computational complexity of said layer. A brief example of how it works follows. With a typical image we have a matrix of width by height by color, where color is a red, green, blue numerical representation (RGB). A typical CNN layer will convolve the image with a kernel over the entire 3D space with one operation such as reducing a 3x3x3 space into a 1x1 space. A depthwise separable convolution uses a kernel whose depth is equivalent to the image depth and therefore would reduce a 3x3x3 space into a 1x3 space. This keeps the convolution separate for each channel and provides separate weights for each channel to learn during training. After this layer, a point wise convolution is performed combining the 1x3 space into a 1x1 space giving another layer for weight training.

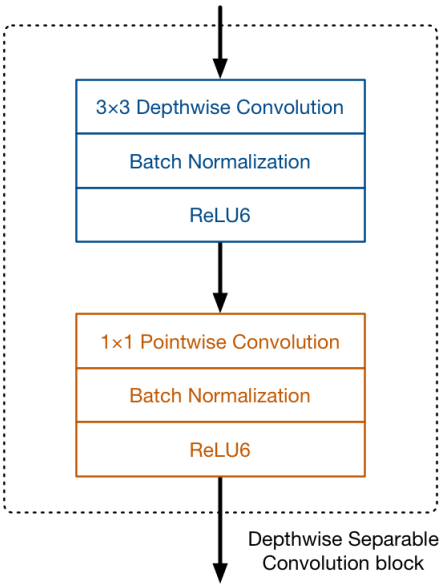



Figure 2.2: This depicts the MobileNet Main Network Element. The data flows from top to bottom reducing the 3x3x3 feature into a 1x1 feature using the ReLU6 activation function [26].

V2 is a rework of the original architecture [24]. The main contribution in version two is the introduction of the inverted residual with linear bottleneck. The goal here is to reduce the amount of channel information being stored and passed around between layers, which trims the amount of memory and multiply add operations needed. To do this, the depthwise convolution from version one is surrounded by first an expansion layer and then a projection layer. The expansion layer expands the features coming in by an expansion factor, essentially a multiplier of the input size. Then the projection layers acts as a bottleneck and reduces the feature space as expanded by the expansion layer. This means the features being learned from the depthwise convolution are of a higher dimension than the information being passed in between layers, essentially compressing the information between layers and saving space.

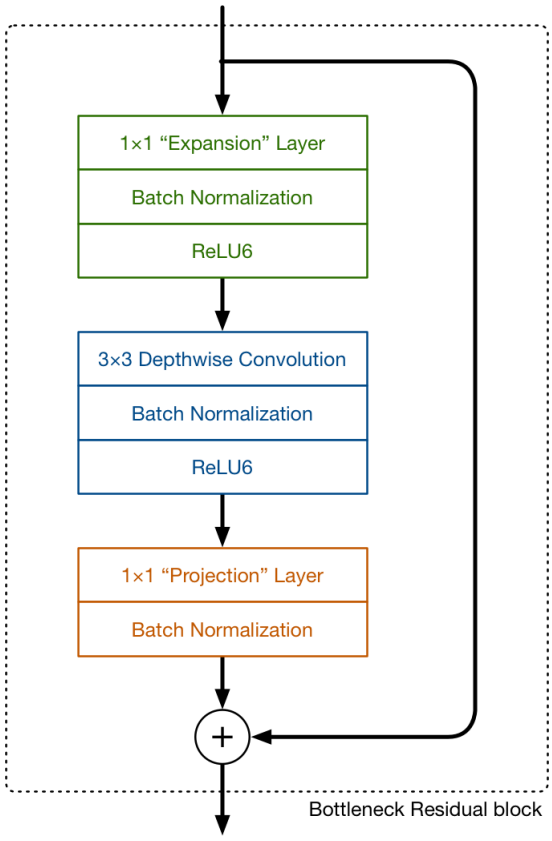



Figure 2.3: MobileNet V2 Main Network Element is a residual block. This block is used iteratively throughout its design and allows the network to understand pixel neighbor relations [26].

SSD [27] is a single shot detector that picks out where in an image a certain recognizable object is. The MobileNet V2 is used as a feature extractor and placed in front of the SSD network which uses these features, tied to locations in the image, to output bounding boxes around identified objects. It works by first creating a set of default boxes setup throughout the image. It then uses the object classifier to score the information within a box and if its recognizable, an object we are training to detect, it adjusts the box size to fit the object. Returned from the SSD is a bounding box, a label, and a confidence value from 0 (low) to 1 (high). Figure 2.4 presents example of this algorithm running on a sample image. The bounding boxes are drawn around the objects and in the top left you can see its label and

confidence.

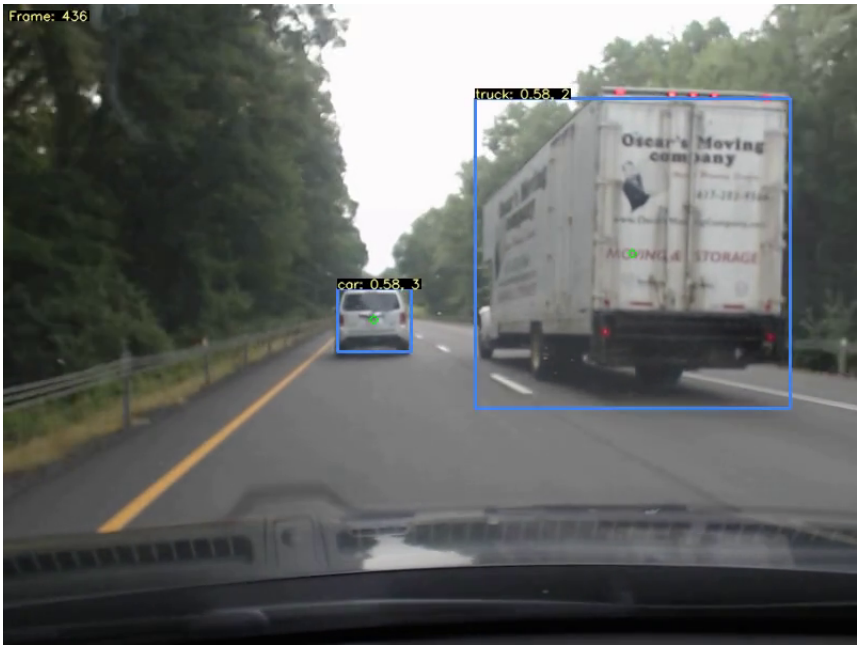



Figure 2.4: Objects being detected are annotated by bounding boxes. This image shows two different objects being detected. One a car with a 58% confidence and the other a truck with the same level.

The main reasons for choosing this architecture is that the tradeoffs between accuracy, speed, and portability fit perfectly with the requirements of this work. To operate on hardware, in real time, we need speed and to keep the cost reasonable we need portability. In addition to performance, there are pre-trained models offered for this architecture. One of which already contains the object classes required for this work. This leveraged model has been trained on the Common Objects in Context (COCO) dataset [28], which includes the classes of car, truck, and pedestrian among many others. These pre-trained models are provided by Google [29].

The performance goals are described in Chapter 3 and plenty of examples were found using this network to operate well within those requirements [30]. There was no real competition in this area for a detection architecture on an embedded platform. For more information about performance, see Chapter 3.

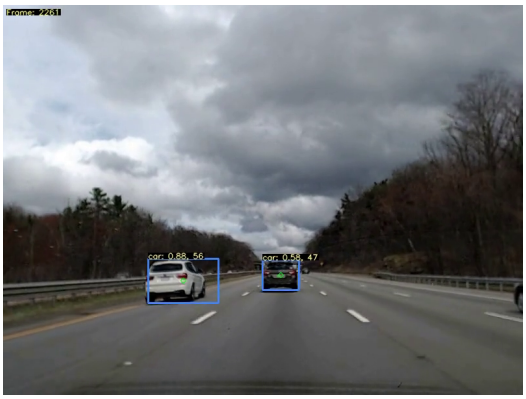
### 2.1.2 Tracking

The trackers used here are to understand the movement of an item between consecutive frames in a video stream. This is more formally known as object tracking. There are many different model types that can be used to solve this problem [31]. One type is a motion model, by using either a Kalman filter or an equivalent predictor to keep track of the motion one can use the objects momentum to predict where it should be in the next frame [32]. Another would be to use an appearance model, using something like a correlation filter, one can build a representation of the objects features (shape, color and size) and filter the image for this representation [33]. Alternatively, for an appearance model, you can use an online classifier which is continuously trained on the object's appearance and used to locate the object in the next frame. In this work, we will be focusing on correlation trackers as they offer a good performance and accuracy trade-off. Although it could be true that a hybrid of these models could work better in both categories, this is outside the scope of this work.

The tracker used is the Correlation Tracker implemented in the dlib library from the following work [34] [35]. This tracker works as follows. It starts with an image containing the desired object to track and a bounding box surrounding the objects location. The tracker uses this to create a feature representation of the object and builds a scale and translation model in a latent feature space (different feature dimensions than the image). This can robustly track changes in the objects scale, which occur often in the self-driving vehicle space as the camera is typically in motion. When the next frame in the video stream comes in, the tracker is called to update the initial bounding box with the object's new location. This is done in three steps. The algorithm first extracts a scale sample from the image and uses a correlation filter to sweep its previous model over the current sample and calculates the correlation across the entire image. It then finds a position in the image which maximizes the correlation. It does the same with the translation model. Finally it updates the translation and scale models with the new positions feature representation. This process is repeated for every new image. An example of an object being tracked between images is shown in Figure 2.5.

The tracker has a few parameters which allow you to adjust the performance and accu-

(a) Track Example 1



(b) Track Example 2


(c) Track Example 3

Figure 2.5: The image sequence highlights the trackers ability to track, not just the objects location, but also its scale from frame to frame. The white vehicle on the left is moving quickly past the camera and this movement is reflected in the area of the objects bounding box.

racy tradeoff. These parameters are for the size of the filter and the number of scale levels. The filter size refers to the pixel by pixel matrix used for the correlation filter. When the filter is large it is more accurate but has a longer computation time. The scale levels are the

levels of detail in the scale model of the feature representation. The same trade off applies here. It also outputs the peak to side-lobe ratio after every update allowing you to know how well it thinks it is tracking the object. The higher the value to more confident it is. This tracker is used out of the box to help bring the project up quickly.

## 2.2 Signal Processing

### 2.2.1 Smoothing

Signal smoothing, as typically accomplished through moving windows, averaging filters, and resampling, helps to highlight trends and anomalies in data by reducing the outliers or dampening the noise. It is often the first step in exploratory data analysis or a signal processing pipeline when looking at real world data.

The goal for smoothing our signals is to find clean oscillations and sudden anomalies. To highlight these trends in our data we want to eliminate higher frequency noise without dampening sudden changes. The proposed system uses a Hanning weighted moving average filter with a small window size relative to the systems target operating rate. This helps to only filter out high frequency noise and retain the subtle trends. The Hann window is a variant of a raised cosine window [36]:

$$w[n] = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{N}\right)\right] \tag{2.1}$$

$$w[n] = \sin^2\left(\frac{\pi n}{N}\right) \tag{2.2}$$

where $n$ is the current index and $N$ is the window size. It weights the samples within the window by giving the largest weight to the current value and tapering off the involvement for the surrounding samples as you move away from the center. This helps to retain any anomalies by heavily weighting the current value and smooths out high frequency noise with the tapering slope to zero, resulting in a window which looks like those in Figure 2.6.

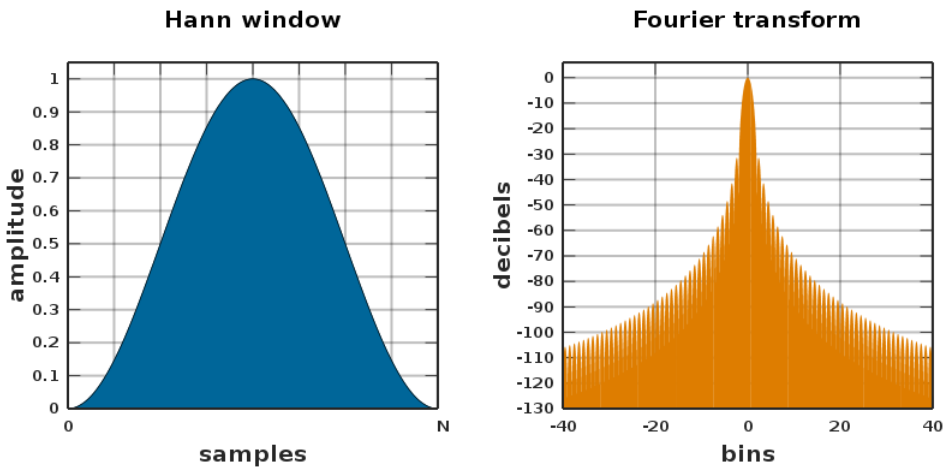



Figure 2.6: Hann Window and Fourier Transform showing how the weights are distributed in the time domain on the left and frequency domain on the right [37].

Figure 2.7 is an example of the Hann window smoothing out a noisy signal with a couple of anomalies. This outlines that our goals as described above are achieved.

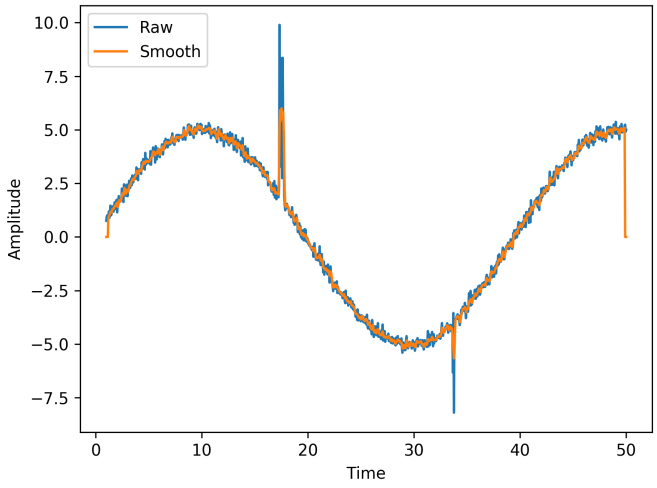



Figure 2.7: This Hann window example illustrates how high frequency noise is smoothed out while dampening outliers, but not completely removing them.

### 2.2.2 Anomaly Detection

The anomaly detector‘s goal in this work will be to highlight unusual behavior in the incoming environmental signals. In particular, the output of this detection will be true when the signal deviates from its typical path within a moving window. Many methods for anomaly detection exist and have different goals. Some look for peaks by curve fitting and calculation of residuals. While others look to match patters like wavelet transformations [38]. Others will leverage forecasting or classification methods to track features [39, 40]. When the features fail to be predicted, fall outside of a common cluster or into new classifications then that point is considered an anomaly.

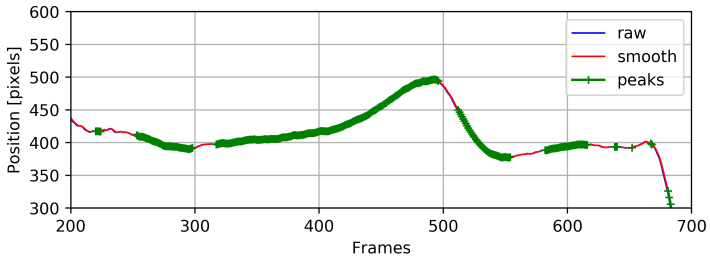
These techniques also fall into two main categories of execution: real time or whole signal processing. Real time gives an output based only on current and past states providing a value as it moves though the time-series as its being generated. While whole signal looks at the entire signal at once, after the signal is generated, and will mark areas of interest at every point using both past, current and future signal values. For this work we are only focusing on real time methods.

The method chosen for this work is simple, but effective for the scope of data we will be looking at. It is known as Robust peak detection algorithm using $z$-scores [41]. It functions by calculating a mean and flagging anything outside of $X$ amount of standard deviations, but with a few extra features which make it more robust. This algorithm is used in similar related work [42, 43].

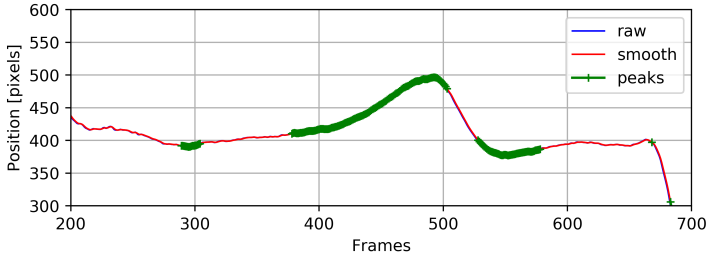
It works by first creating a moving window, known as the lag. The lag specifies how many samples are used to calculate a filtered mean and standard deviation. A threshold is then set to choose the amount of standard deviations the signal needs to be away from the filtered mean before the detector is tripped. In addition to the threshold an influence parameter is also set. This lets the threshold know what percentage of the incoming signal can influence the detection conditional. In other words, an influence of one means it disregards the filtered values when calculating the threshold and a zero means it only used the filtered values. The influence parameter assists with sensitivity to trends in the data. A value of zero assumes that the signal is non-stationary and would respond erratically if the trend, relative to the

window size, change quickly.

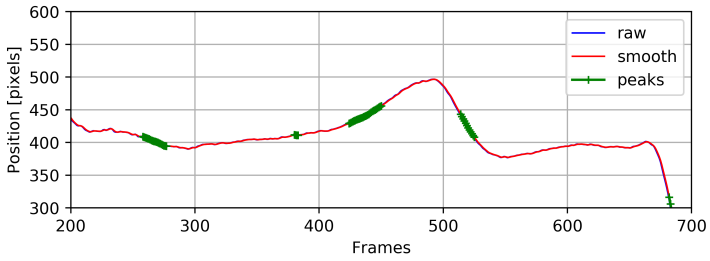
Figure 2.8 is an example of the algorithm in action on a section of collected data with various changes to the parameters:




(a) Threshold of 1.0, influence of 0.9 and lag of 37 frames




(b) Threshold of 1.0, influence of 0.9 and lag of 120 frames




(c) Threshold of 2.0, influence of 0.9 and lag of 37 frames

Figure 2.8: The parameter changes above help to highlight the algorithms ability to select certain parts of a peak. However, the more of the peak selected the more susceptible it is to noise.

This helps to highlight the many ways that this algorithm can be used depending on the part of the signal you are interested in detecting. For this work we would like to know when the large hump starts to occur and ignore small fluctuations in the signal.

### 2.2.3 Oscillation Detection

The purpose of the oscillation detector is to find movement in the signal represented by low frequency fluctuations relative to the high frequency noise. These fluctuations of interest can be detected using a fast Fourier transform. An FFT is a common technique in signal processing which will transform the signal from a time-based representation to a frequency based one. Instead of a signal which is represented by amplitude verses time it will be represented by amplitude verses frequency. The frequencies of interest are known and to detect these fluctuations one can simply view the energy level of that frequency in its frequency representation [44]. Figure 2.9 is an example of an FFT.

For this work the oscillations that we are interested are in the last two images and can be detected by looking at the energy in bins 2 though 5.

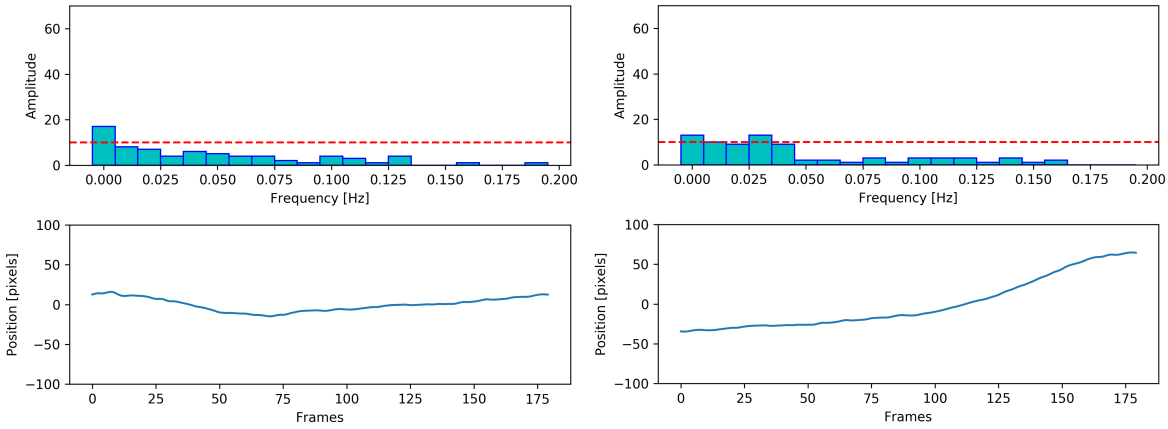
## 2.3 Machine Learning

### 2.3.1 Feature Engineering

In between data collection and algorithm application there is an important step which helps fit the data to the problem being solved and to the algorithm being used. Feature engineering is not used only to make the dataset compatible with the machine learning algorithm, in syntax and variable typing, but it helps to give the algorithm a hint as to what it is trying to learn. Whether it is removing irrelevant information or applying post processing techniques the point is to increase the accuracy of the task.

The task is to group similar movement from time series signals. One option when attempting this is to use the signal directly as the only feature, but it is typical to derive sub-feature from this signal which aim to describe the types of behavior that you wish to group. Time series sub-feature decomposition is a large field often explored by economists for the purpose of stock market prediction. The signal movement to be grouped is like that of the stock market and "catch22: CAnonical Time-series CHaracteristics" covers an extensive exploration of feature extraction for that purpose [45].

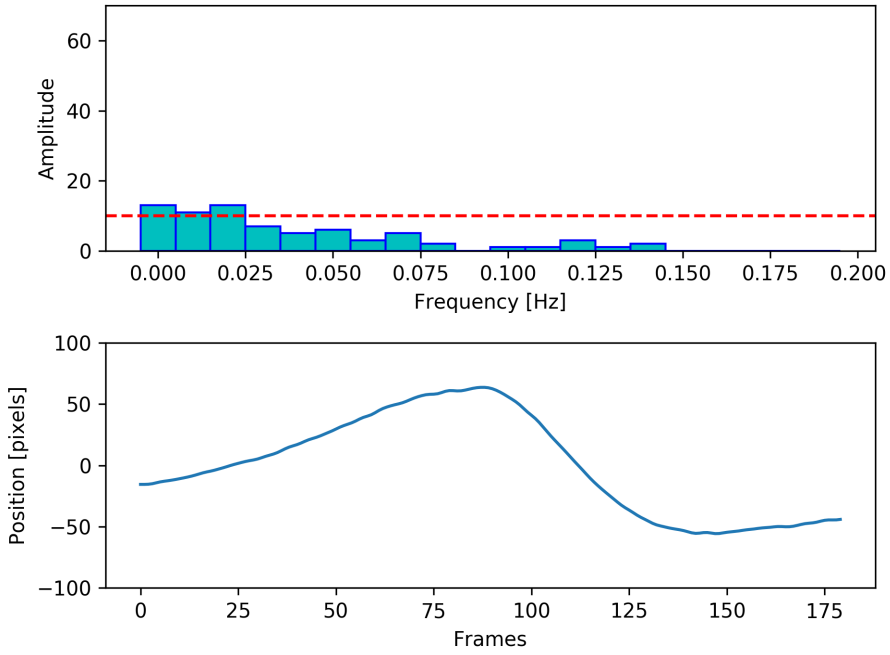
The objective is to find the most representative sub-features of a time series signal such




(a) FFT of a flat sigal




(b) FFT of an upward trend




(c) FFT of an oscillation

Figure 2.9: Various movement can be seen in the time series signals above. This image sequence highlights the changes that the systems aims to detect. A progression from flat to an oscillation

that they contain no redundant information and work well over many types of applications and datasets. Specifically, the work targets 93 classification datasets for analysis. The

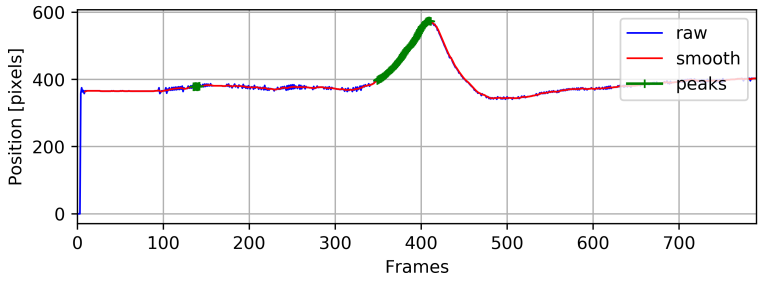



Figure 2.10: This is an example signal used to show what each of the features look like in the catch22 set.

features to be tested are from a common feature engineering toolbox known as "hctsa" which contains 4791 features [46]. To find this "catch22" subset, the authors apply all these features to the classification tasks for each dataset and filter by performance and within the top features they reduce redundancy by ordering the cluster feature correlation matrix. This last step is similar to how principal component analysis works [47]. An image below from the paper clearly illustrates the process:

For this work, the catch22 subset of features is leveraged as the basis of our time-series feature decomposition. In Chapter 4, we will be exploring and adjusting this base feature set. Figure 2.10 is an example of the features being used on a time series signal.

### 2.3.2 High Dimension Visualization

When working with a dataset whose feature size is outside of the visual domain, with sizes larger than three, it can be difficult to understand how the data points are related. When dealing with data inside this range, you can plot the various features on a 2D or 3D plot and see relationships in the data. The only way to find these relationships visually with higher order data is by using dimensionality reduction techniques. In this work, t-distributed stochastic neighbor embedding (t-SNE) will be used [48]. Unlike another common technique, principal component analysis (PCA), which relies on linear relationships in the features, t-SNE aims to find nonlinear relationships [47].

t-SNE's aim is to take high dimensional data and create a mapping for it in a lower

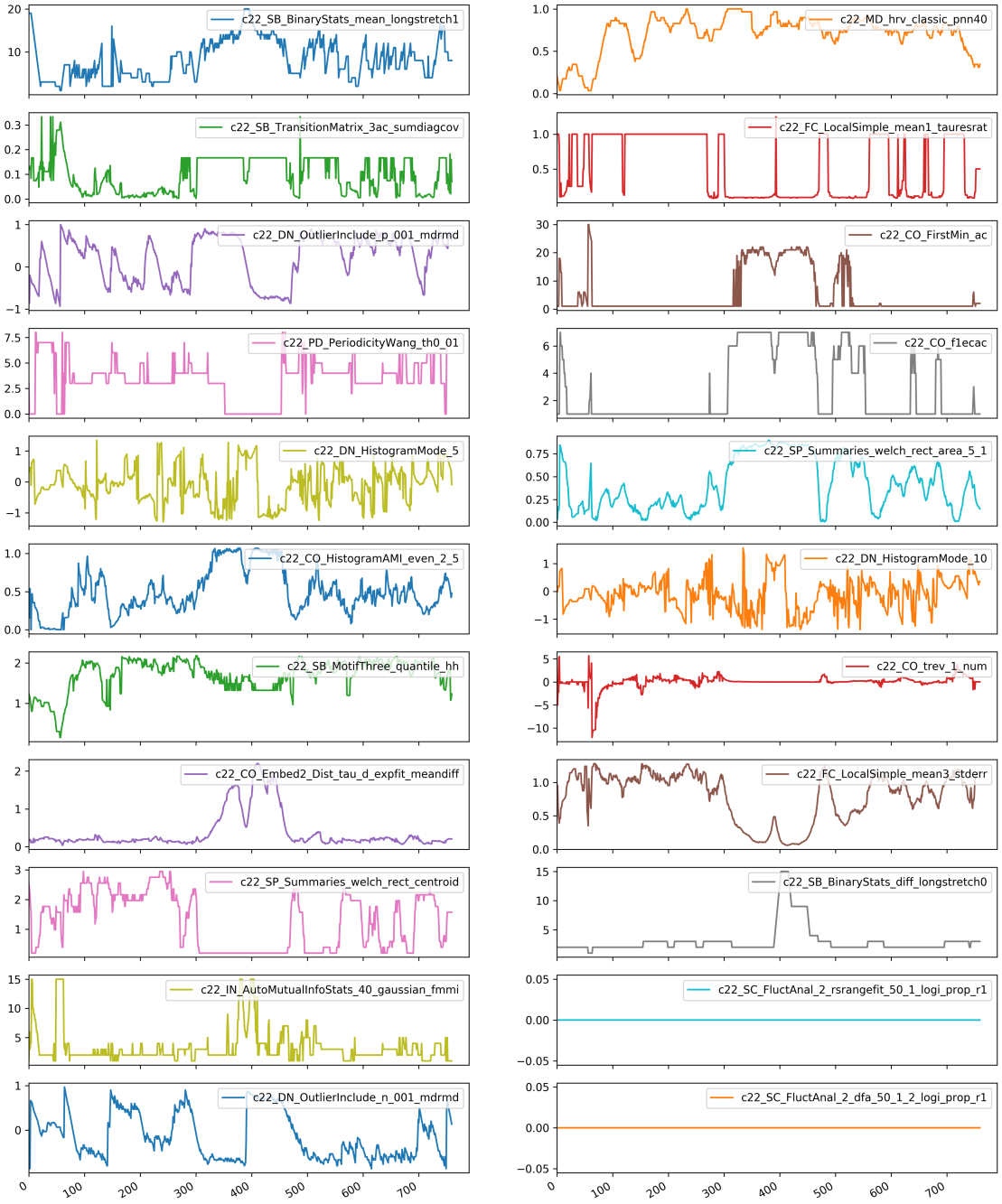



Figure 2.11: Each of the 22 features is created by moving a window through the signal and calculating each feature on the windowed portion data.

dimensional space though probabilistic relations. It first looks at the input features and calculates, for each point, its relationship to every other point. This relation is then fit to a normal distribution curve that is created based on each input features group density. It then projects the input feature points randomly onto a lower dimensional space and calculates the relations again between points using a t distribution. The final step is to iterate along a gradient that minimizes the difference between the similarities calculated in the input space to the ones calculated after the projection to the lower dimensional space. The result is a representation of the input features which are grouped attempting to retain the neighborhood relations of the original space. This allows you to visualize higher dimensional relations. For an example of how this is used, see the next section as it is best understood in its relation to clustering.

These tools are used in this work to help us understand the relationships of the different data points in our high feature space. They are not used during runtime, but rather for analysis.

### 2.3.3 Clustering

Clustering is a powerful technique that can group together datapoints which share similar features without any labeled data. Four main paradigms exist for clustering. Hierarchical methods break down the relationship between points though an ordered structure where all points belong to a top tier and a subset of those belong to a lower tier and so on. One example would be agglomerative clustering [49]. Partition based clustering create borders between groups either with line fitting or by assigning them to a modeled distribution type. One example would be k-means clustering [50]. Density based clustering find groups according to the closeness of data points and separates them by either differences in density or differences in group location. One example would be ordering points to identify the clustering structure (OPTICS) [51]. Grid techniques break up the space into pre-defined grids and assign groups to each one making use only of the grid boarders for separation. One example would be statistical information grid-based method (STING) [52].

The result of clustering is when each data point is assigned directly to one group or to multiple groups each with an assigned probability. This is known as hard or soft clustering

respectfully. For this work, the clustering algorithm will be used to classify sub-features of a time-series stream of information into hard clusters. The feature count will be larger than three and not directly plottable. Also, due to its streaming nature it is likely that points in time could be close together and the border between groups may be difficult to distinguish. In addition, it would be helpful to avoid re-clustering all previous datapoints seen for each new datapoint added as the signal continues through time.

OPTICS would be a good choice in this case as it is a density-based algorithm which expects each cluster to be able to vary in size, density and location. It also does not require every point to be in a group and can have the tight border points remain unlabeled. However, OPTICS is significantly slower than the algorithm is it derived from [53]. DBSCAN works well when clusters are similar in density, which in this case is seen as a disadvantage compared to OPTICS. Both algorithms do not require the number of clusters beforehand, the only parameters have to do with density and separation which help choose the cluster borders. The automated nature of these clustering techniques can be helpful in this work.

However, when it comes to performance and accuracy trade off it is hard to beat mini-batch k-means [54]. K-Means iterates along a gradient set to minimize the error as calculated by the squared distance between every point and its nearest cluster center. The mini batch variant allows points to be brought into the clustering space iteratively by splitting the incoming data into batches. This allows it to run in real time and prevents the need to re-cluster previously seen points.

Exploration was done for many different algorithms. Please refer to the appendix B.1 for the accuracy and performance comparison. Figure 2.13 is an example of the clustering algorithm applied to the time series subfeatures as shown above and the dimensionality reduction technique used to display the results on the 2D plane.

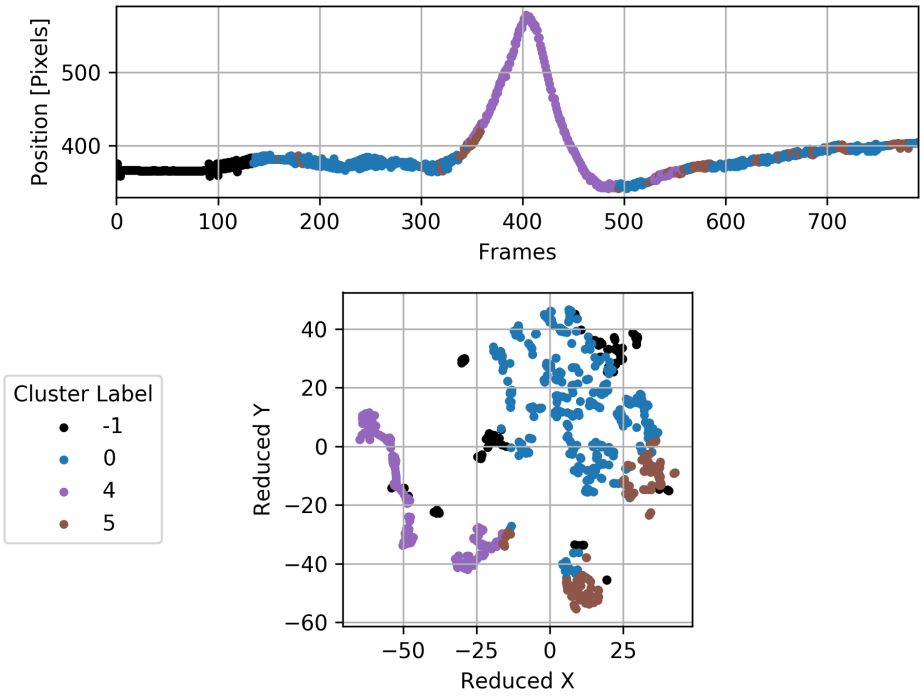



Figure 2.13: The top plot is the signal used in the feature engineering section except it is now colored by the time steps corresponding cluster. The bottom plot are the features plotted on the 2D plane using t-SNE and also colored corresponding to thier cluster.

## 2.4 Summary

This chapter reviewed the processing methods used to help solve the problem of identifying distracted drivers. This work is borrowing techniques from three different disciplines namely, image processing, signal processing, and machine learning. The goal is to use expertly defined algorithms like those often found in signal and image processing to preprocess data and to help frame the problem such that an unsupervised clustering systems results can be interpreted and immediately useful during runtime. The age of the topics here varies, but all have been heavily used in practice and would be considered robust.

Chapter 3, will review the hardware and software and how these methods will fit into each part.

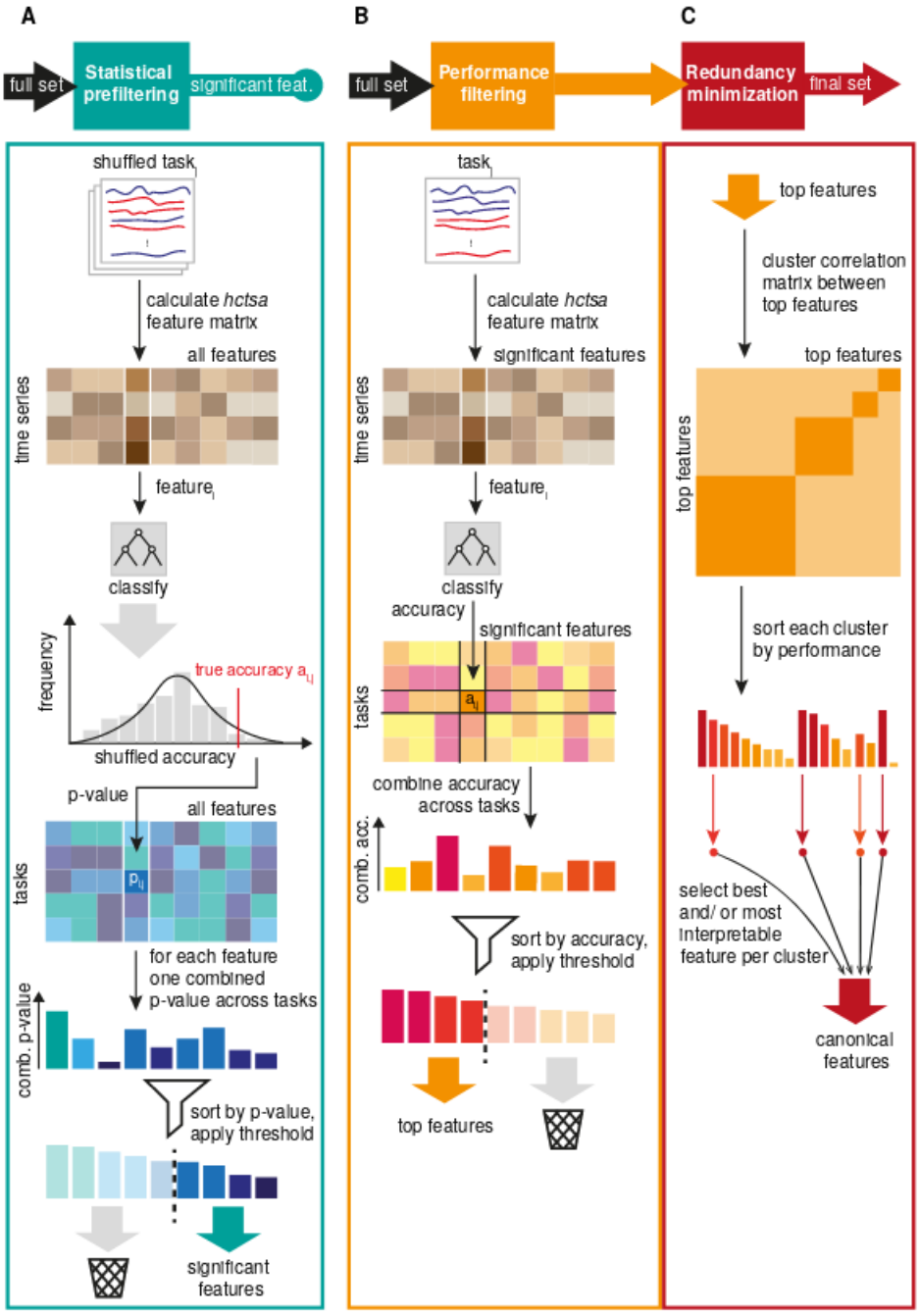



Figure 2.12: This workflow was used for the creation of the "catch22" set of sub features. The first step was to test each one out on several datasets. Then rank them and eliminate the ones which hold redundant information [45].

# Chapter 3

# Hardware and Code Architecture

## 3.1 Hardware

The goal of the hardware is to execute the processing pipelines described in Chapter 2 while meeting certain performance criteria. The goal for this work is to be able to process live video at a resolution of 800 pixels wide by 600 pixels high at 30 frames per second (FPS). The hardware was selected to meet these goals while also optimizing for cost. The processing modules chosen could be purchased at bulk and at a reduced cost, where all parts could be combined onto a standalone circuit board.

Aside from processing constraints, the hardware module must function standalone while working with a standard vehicle power supply, handling the video input and informing the driver of the distracted vehicles. A detailed diagram of the hardware is shown in figure 3.1, and its components will be described in the following sections.

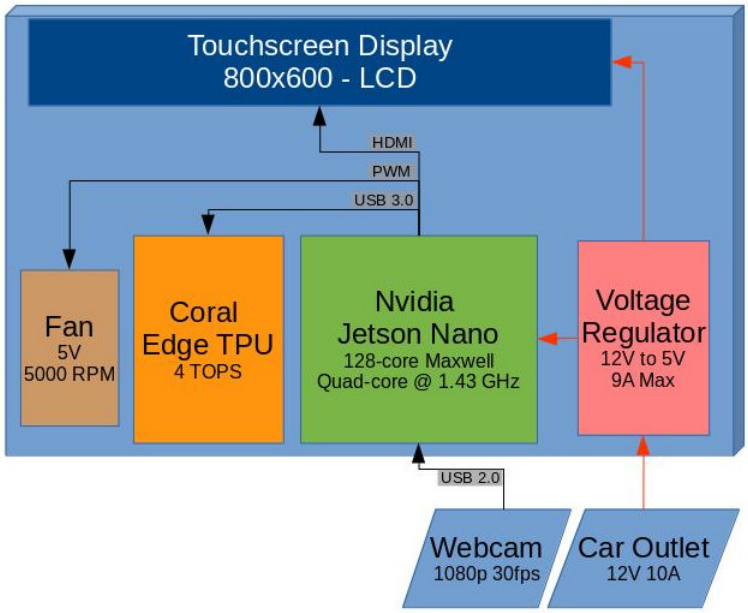



Figure 3.1: This block diagram represents, by location, the hardware components that have been mounted for the in-car unit.

### 3.1.1 Main Control Board

The main performance bottleneck for this project is the image processing section. Here we must detect the vehicles, track their movement, and mark the lines on the road. Many embedded boards can run these algorithms, but with poor FPS and/or resolution. Some higher cost board also exist with the processing power but would make this work less accessible. The two boards who stood out as potential candidates after the previous considerations were the Google Coral TPU development board and the Nvidia Jeston Nano. These both have hardware acceleration for the detection task, however, the TPU would only be able to accelerate methods based on deep learning with a limited set of neural network building blocks. The Jetson Nano has a 126-core graphics processing unit (GPU) which could be used for the acceleration of all image processing tasks this work requires, but not at the same time. Table 3.1 describes the boards considered and their processing power [55–60].

Table 3.1: Development board comparison between the top contenders in the affordable edge AI space.

| **Board** | **CPU** | **Acceleration** | **Performance (MobileNet v2)** | **Cost** |
|---|---|---|---|---|
| Coral Edge TPU | ARM A53 (4x 1.8 GHz) | Edge TPU | 100+ FPS | $149.99 |
| Nvidia Jetson Nano | ARM A57 (4x 1.43 GHz) | 128-core Maxwell | 64 FPS | $99.00 |
| Nvidia Jetson TX2 | ARM A57 + NVIDIA Denver | 256-core Pascal | 17 FPS | $399.99 |
| Raspberry Pi 3 | (4x 1.2GHz) | N/A | 2.5 FPS | $35.00 |
| Intel Neural CS2 | N/A | Myriad X VPU | 30 FPS | $90.00 |

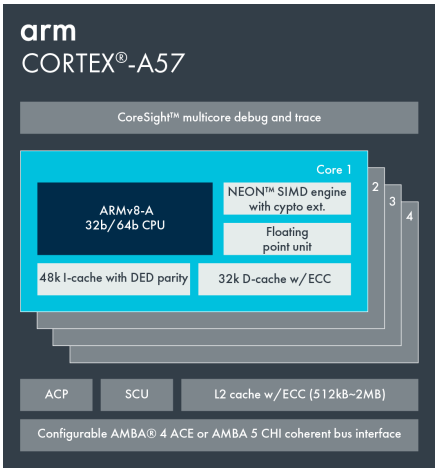
Considering the options for a development board and other acceleration hardware available on the market, a hybrid approach is the best way to go. Aside from the Coral development board, Google also offers the TPU acceleration as a USB 3.0 module allowing it to be used on any development board capable of running Linux. After this consideration the choice of the Jetson Nano became clear as we could leverage the GPU to accelerate many of the image processing tasks and have the Coral TPU USB version dedicated to the detection task. More information about TPU and GPU acceleration is described in the following sections.

The Edge TPU was developed to help bring machine learning to embedded devices with excellent power efficiency. It is targeted towards image and audio processing projects, but it can be used for a variety of tasks. It does not provide acceleration for training a network, but only for inference, as it has no acceleration hardware for backpropagation. It can, however, provide acceleration for retraining pre-trained models.

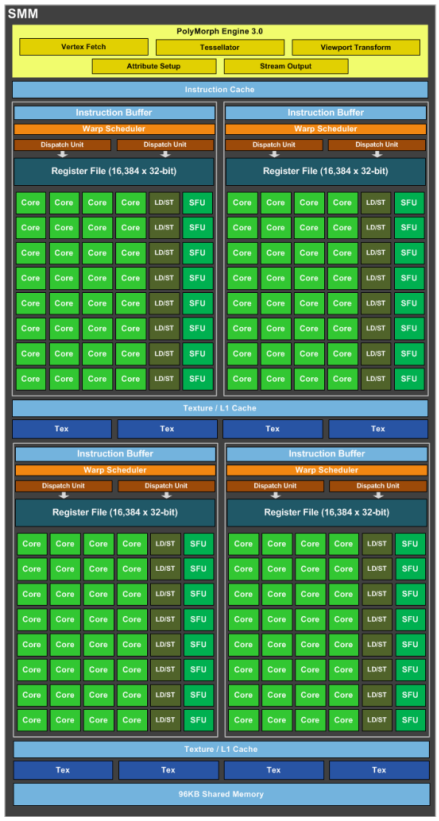
The Edge TPUs architecture is not released at the moment, but it is an application-specific integrated circuit (ASIC) that has been created to optimize the inference of deep feed-forward neural networks (DNN). It can perform the necessary feed-forward operations, quantized to 8-bits, at 4 trillion operations per second (TOPS). Typical DNN operations include rectifier (ReLU), tanh, add, multiply, 2D normal, and depthwise convolution, Max Pooling, Softmax and Mean among many others [61]. With function specific hardware, and when massively parallelized, large performance gains and computational efficiency takes place as seen from the examples. Not to mention that when offloading such complex tasks

to the TPU you free up the GPU and CPU for other work. For this work, the object detection task will be handled entirely on the TPU. See Chapter 2 for more details.

GPUs are versatile in the tasks they can accelerate. Any task that can be performed in parallel is well suited for GPU acceleration. Figure 3.2 is a comparison image of the CPU and the GPU architecture we are using. You can see that GPUs are built for parallel computations and many tasks in image and signal processing can take advantage of that. Unlike the TPU, the GPU is not restricted by operation type or direction and can be used to accelerate learning as well as inference in a DNN.




(a) Cortex A57 CPU [62]




(b) Maxwell GPU [63]

Figure 3.2: The image on the left shows the core components of the A57 CPU, which has 4 processing cores sharing one L2 cache. The image on the right shows the many GPU cores of the Maxwell GPU. These are split into 4 sections each with its own cache.

For this work, we are looking to accelerate an image feature tracker, a lane segmentation

network and our custom distracted driver network. For more details, see Chapter 2.

A USB 2.0 webcam will be used for our video stream input. Any webcam that can meet the performance requirements will do, as it can be meet for a relatively low cost with the webcams available today. The main parameters to look at when considering a webcam for self-driving car work is resolution, FPS, and viewing angle. The viewing angle and the resolution interplay are important to understand as when the viewing angle of the camera increases the pixel per area decreases and view distance is lost. For this work we are only considering cars in front of us on the road and dont need to see sidewalks or cars approaching from adjacent roads so a smaller angle will work. The FPS is equivalent to the sampling frequency in hertz. To detect distracted driving, we must capture at a rate which retains any side-to-side and acceleration movement that a car can make. For an average car we can safely capture and view this at 30 FPS as the typical lane change time of a car is 1.5 seconds at highway speed [64] and the recommended FPS to capture traffic movement is 10 [65].

The webcam of choice for this work is the Logitech C920 [66]. This was chosen for two main reasons. First, the webcam has onboard H.264 video encoding for up to 1080p at 30 fps, which offloads some of the work from the main board for video capture. Second, this webcam has been used in other self-driving car image processing work. Part of that work included heat endurance testing to prove that this camera can hold up in a car on a hot day and not degrade performance or melt. In addition, the viewing angle of 90 degrees proves to offer a good width to distance trade off.

### 3.1.2 Power and Heat Managment

The power supply from a vehicle is at 12 volts and typically capable of supporting up to 180 Watts (15 Amps). The total power requirements for our hardware module is shown in the table 3.2.

A voltage regulator was chosen to support the maximum power requirements with some overhead. This voltage regulator is created by Pololu part number D24V90F5 and can convert 12v to 5v with 90% efficiency and a 45W (9A) output.

The major sources of heat in our hardware module are from, in decreasing order, the

Table 3.2: Total power broken down for each component in the hardware system.

| **Part** | **Volts / Amps** | **Power (Watts)** |
|---|---|---|
| Jetson Nano | 5 / 2.2 | 11 |
| Coral TPU | 5 / 0.5 | 2.5 |
| Webcam | 5 / 0.5 | 2.5 |
| Display | 5 / 1.2 | 6 |
| Fan | 5 / 0.5 | 2.5 |
| **Sum** | 5 / 4.9 | 24.5 |

TPU, Jetson Nano, and the power supply. This order was found through performance load testing of all components and manually measuring the heat differences between the components by hand. The Jetson Nano and the TPU come with heatsinks, but as we are sandwiching them between a poly-carbonate backing and a display a fan must be added for proper airflow.

The Jetson Nano has temperature monitoring and has a pulse width modulation (PWM) output to support common computer fans. The fan selected came from the development guide recommendation. It is manufactured by Noctua part number NF-A4x20 and runs at 5000 rotations per minute (RPM). The fan improved heat buildup dramatically, as shown by running the same performance load test as before and comparing the temperature difference.

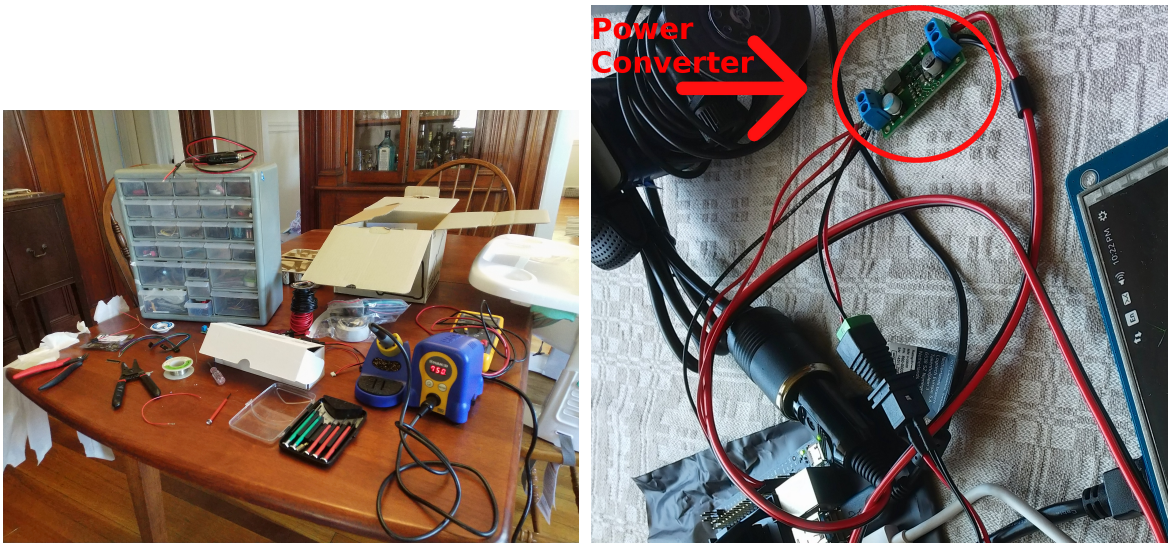
Several test runs have also been performed while running the system in a car to ensure the heat is kept low and that the power supply is enough at its maximum load.

### 3.1.3 Integration

Two components for the hardware portion must be assembled before operation. The first is the wire harness to supply power to all the components. The second is the support structure which encases the hardware and supports mounting to the vehicle.

The wire harness must make three connections. One from the cars power outlet to the voltage regulator board. Then two, from the regulator to the development board and the

display. The board is capable of drawing 6 amps and the display is capable of drawing 2 amps. To support the current loads going to the dev board and display, 22-gauge wire was chosen and multiple wires were ran to the connectors to support the maximum current draw. A car power outlet pre-assembled wire harness was purchased to handle the connection between the car and the regulator. This is already at the proper wire gauge and has a built-in fuse and power indicator. Figure 3.3 is an image of the assembly setup and the completed wire harness.

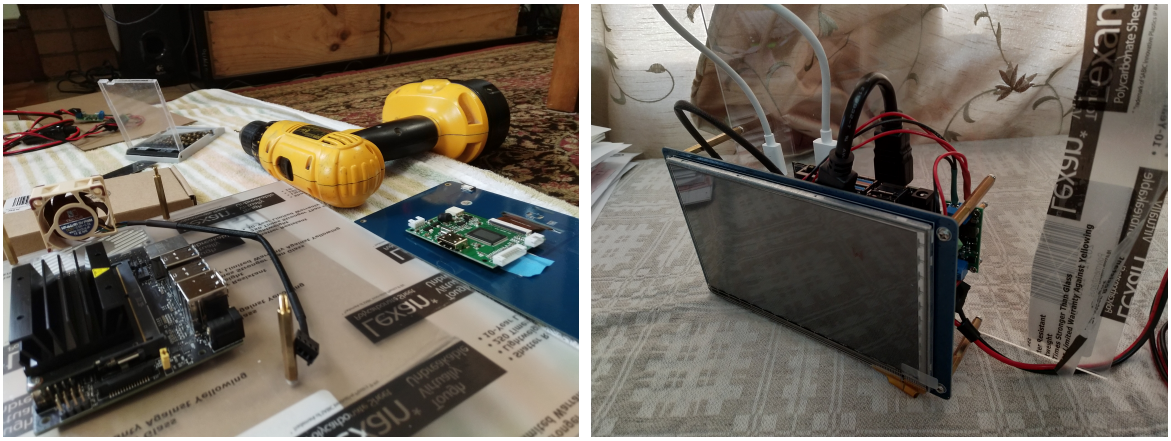



(a) Assembly Setup (b) Completed Harness

Figure 3.3: The setup for this project required a soldering iron, heat shrink, solder, and wire strippers. Once assembled the power harness can split the power coming from the power converter to the display and Nvidia board.

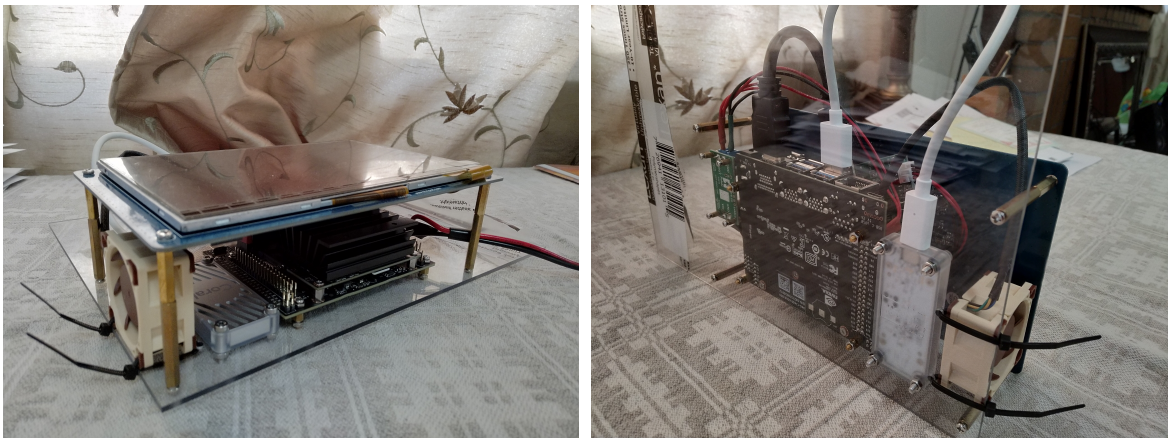
The support structure for the hardware must compactly fit into a car and be able to be secured against movement and vibration. The display should also be visible. The camera and power cords must be able to reach the unit as well, but this is dependent on the vehicle's setup. The simplest design, and the most efficient, is to sandwich the hardware components in between the display and a polycarbonate backing. Polycarbonate was chosen as it is far stronger than Acrylic and could be thinner to support the weight needed to mount of the components. There is no need for the clearness of Acrylic. All components are secured with

appropriately sized machine screws and nuts as required for each part. The only drilling required is in the backing to thread through the screws. The placement of the components can be seen in figure 3.1 at the beginning of this chapter and are in order of heat production and considerations of air dispersion. The order looked to optimize laminar flow before the air hits the dev board heatsink and becomes turbulent. Starting with the fan, the components were mounted left to right including the TPU, Jetson Nano, and voltage regulator. Long brass standoffs are used to secure the display to the backing after the mounting of the components. The result is a ridged rectangle encapsulating all components with adequate air flow and vibration resilience. Figure 3.4 are images of the assembly and final product.



(a) Assembly Setup (b) Completed Front

Figure 3.4: The assembly processes mainly involved drilling holes and aligning the various components. Three different screw sizes where used and the power drill made quick work out of the polycarbonate.



(a) Completed Bottom (b) Completed Back

Figure 3.5: The different angles provide a look at each component and show the completed design.

The setup will be different for each car. The car used to support this work is a 2013 Honda Accord and the unit can be mounted right over the car's info display system. The in-car setup is pictured in figure 3.6 with the system up and running.


Figure 3.6: Here the system is running in real time. The car seen through the windshield is the same one being tracked in the display and a few other objects are detected as well.


Figure 3.7: Circled in red above are each of the connections needed for system operation. From top to bottom the items shown include the webcam, assembled system, and the power connector.

Throughout testing and demonstrating the hardware it became apparent that the need for a simple display was necessary. The live streamed video was too distracting and the bounding boxes around the vehicles are moving around too much. This work acknowledges the need for a simple display like this and has designed a prototype. However, it has been deemed outside the scope of this thesis to integrate it into the system. The goal of such a display is to quickly communicate information to the driver about the surrounding vehicles just as a GPS would do.

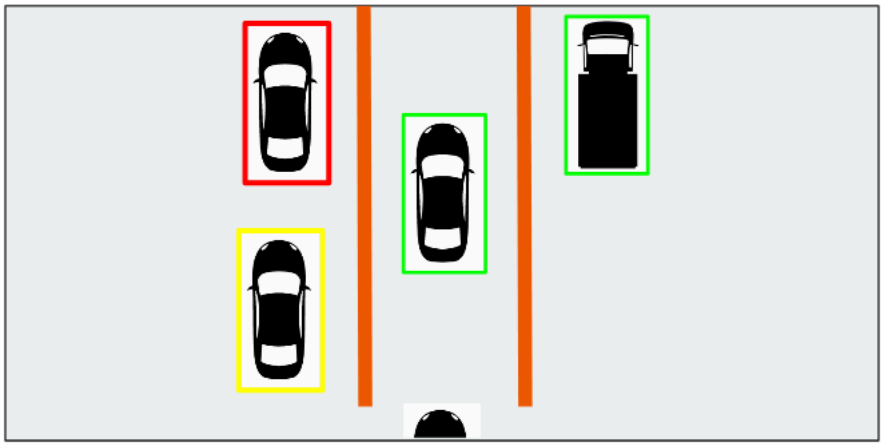


Figure 3.8: This is a simplistic representation of the outside world which aims to delivered the distracted vehicle information as fast as possible. Icons provided by [67].

The location of the cars will be set relative to their real location and the colored bounding boxes will be drawn to indicate their distraction level. The hood of the vehicle with the system installed is always displayed in the same location and this same style can be extended to add side and rear-view cameras.

## 3.2 Software

The codebase for this project is entirely in Python and using only freely available libraries (C.4). Its main objective will be to act as a catalyst for the projects processing pipeline. Many different methods and functions are fused together to achieve the goal of this work and the details are explained below.

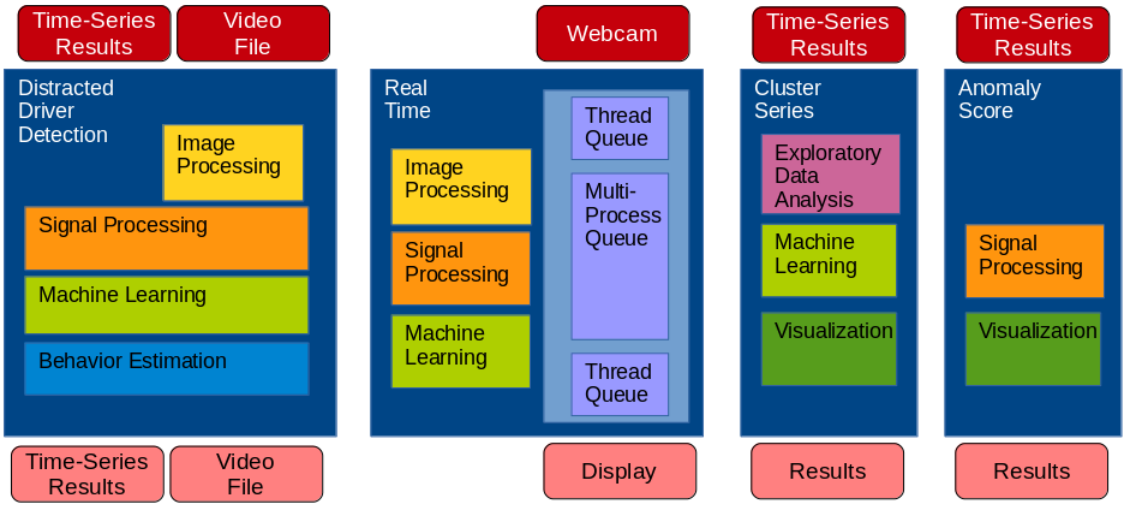



Figure 3.9: These are the main 4 files of the project. This structure allows for piecewise execution of each component to reduce iteration time and ease debugging and tuning.

### 3.2.1 Organization

The codebase is comprised of two main components and two subcomponents, all of which are executable. The separation allows for quick iteration of the contained method. After each execution, intermediate data is saved off to allow for piecewise troubleshooting. Each part is outlined below.

The main class Distracted Driver Detection (DDD) contains the initialization of all major components including the detection, tracking, feature generation, signal processing, machine learning and behavior estimation. When executed it will take in a video file and output the annotated video file with the distracted vehicles highlighted. It will also output a pickle file containing all tracker information and anomaly detection results as a time series pandas dataframe. An alternative mode of execution allows it to take in that same pickle file for plotting and behavior re-evaluation.

Real Time (RT) is the real time script that interacts with the camera and handles threading of all input output (I/O) components. When executed it will pull images from the attached webcam and run the entire pipeline from detection to behavior designation then output the annotated video stream in real time. It contains a queue structure around the

threaded components such that high I/O latencies wont slow down the processing pipeline. Cluster Seres contains initialization of all major learning components and when executed will take in the same pickle file as described above and execute the learning pipeline along with extra visualizations of the process and result. Anomaly Score contains the main signal processing pipeline and when executed will take in the same pickle files and output visualizations of the process and a new re-evaluated pickle file with any updates to the result.

### 3.2.2 Post-Processing Test Bench

The goal of the test bench is to provide an efficient way to test and tune the algorithms without having to worry about real time performance. This process is governed by the playback of a video file that will pause after each frame to ensure all functions complete before continuing. In addition, there are extra graphs and process debugging steps that can occur throughout the duration of the video playback. This allows playback to be paused and the internal state to be visualized at any point. A typical troubleshooting scenario follows.

Choose and playback a video file though the main class, DDD. This class will provide us with the annotated video file, a saved dataframe of all features, and graphs of intermediate variables and decisions. A bug is found where it looks like, in the annotated video, the vehicle is being flagged as distracted when it is simply driving straight. We can look at the graphs to see which feature it thinks is acting irregular. This is decided by the anomaly detectors which reside in the AnomalyScore class. Then, we can make the code changes and replay the dataframe file back through the anomaly scoring function to test the changes without having to reprocess the video (processing the video is the slowest part of the pipeline). The problem has been fixed and the main class can be executed again to re-annotate the video.

The same procedure can be done with issues both in the anomaly scoring and cluster series sub-processes. This saves iteration time and allows testing of new ideas quickly.

In addition to piecewise function execution the main class, DDD, can take in labeling information and output a confusion matrix and other metrics based on how DDDs decisions differ from the label. The labeling in this case is subjective as in most cases we do not know

Table 3.3: Overview of all files in the source directory of the project, how the are used and how they can be configured.

| **File** | **Class** | **Purpose** | **Parameters** |
|---|---|---|---|
| AnomalyScore.py | AnomalyScore | Executes anomaly and oscillation detection for each feature and calculates a score based on distraction impact. | Window sizes, standard deviation thresholds, score values, FFT bin sizes and limits |
| BehaviorEst.py | BehaviorEst | Assess driver behavior based on anomaly score and signal classification. | Designation conditionals, window times and state delays |
| DisplayHelper.py | DisplayHelper | Threads camera and display process; controls video saving and streaming. | Video size, FPS |
| ObjectDetect.py | ObjectItem, ObjectDetect | Helper class to store information from object detector. Sends out image to TPU for object detection. | Detection confidence threshold, model type, detection amount |
| ObjectFeatures.py | HistArray, ObjectFeatures | Array helper function to automatically buffer and remove old information for running live. Extracts features from image and detection results. | Feature list, feature smoothing window and weights |
| ObjectTrack.py | ObjectTrack | Spawns tracker to monitor objects if the detector has trouble. Associates objects from detector to tracker and creates a unique identifier for each one. | Max tracker amount, minimum tracker error, frames to forget an object |
| ClusterSeries.py | ClusterSeries | Runs clustering on time series features to group similar vehicle movement. | Cluster amount, Cluster score, sub feature window |
| Zscore.py | Zscore | Filtered anomaly detection algorithm which automatically handled inputs from multiple signals. | Window sizes, standard deviation thresholds |

what the driver is really doing. It does still provide us with valuable results and give us a baseline accuracy to compare against when any code changes are made.

The visualizations used for troubleshooting accuracy and performance of the system are displayed and described below.

### 3.2.3 Realtime Processing

The real-time processing done for this project is handled by a separate class which favors performance a little bit more than accuracy. As Python is the language of choice, extra work is needed to stay within our performance goals as defined in the hardware section. The sources of difficulty come from the correlation tracker, all IO interaction and the feature clustering. Several different methods exist for tackling these issues. Some of which are optimizations, and some are compromises.

For optimizations code can be parallelized with either multi-thread or multi-process. In Python, these mean different things than in C++. Multithreading spawns a thread within the same process, it shares memory and can execute code concurrently, but not in parallel [68]. Multi-Processing spawns an independent process, it does not share memory and it can execute code in parallel on separate CPU cores [69]. Optimizations also include parameter tuning of the used algorithms. This would ideally yield the same or comparable performance while decreasing run time or complexity. Algorithm changes can also be made which could include vectorizing calculations or changing the order of operations to take advantage of hardware or decisions. For example, one could place a conditional before a set of calculations such that a decision is made and can abort the function before processing time is wasted on the calculations.

For compromises one can choose to drop data or reduce calculation precision in favor of execution time. These all come at the cost of reducing performance. Processing every single frame will not always affect the result as much as one would think. Dropping a frame here or there to keep up with real time might be more important. Other compromises could include reducing bandwidth (sampling time or resolution), using less accurate models or algorithms and skipping functions. There are tricks that you can play with order and time multiplexing in constrained environments. For example, certain algorithms or functions can

be executed interchangeably such that algorithm A would run on even frames and B would run on odd frames. Thinking judiciously about where you spend you processing time will help such tricks and compromises become more apparent. In this work almost all the above methods are used to help meet the performance objectives. The following is a discussion of a few optimizations and compromises made.

The first example covers how to deal with the IO interaction. A solution is needed that can concurrently grab data from an external source such that the processing pipeline does not have to wait for the IO latency time. Since threads share memory and this is a high bandwidth operation, they are the best option. A queueing structure is setup between the IO interaction of both the USB webcam and the Coral edge TPU. Doing this moves the frames per seconds up from 35 to 60 (with no tracker running) which gives us significantly more overhead to work with.

The correlation tracker is the most computationally expensive operation and does not need to be ran every frame. Especially if our detector, which has no trouble running every frame, makes a detection on an already tracked object. If this happens the system knows where the object is in the frame and does not need to perform any additional steps to find it. To solve this an algorithm optimization can be combined with a data-based compromise. The correlation tracker will be setup in a separate process using Multiprocessing with surrounding mutex queues. These queues will then only be filled with a frame if the object association code is missing an object from the detector. When no objects are being tracked the system runs at 45 FPS (with the detector running), detections or no detections. Once the system has the need to run the correlation tracker the FPS will drop to around 15 to 35 FPS depending on the number of objects and the difficulty of any occlusions. This is an improvement to the consistent 12 FPS drop previously.

The last example is about the feature clustering portion of the code. The best clustering algorithm for the features would be OPTICS, although this algorithm is slow. A compromise must be made for a different clustering algorithm. K-Means Mini Batch is less accurate but allows the addition of new data points in real time without the need to re-cluster the past points. The loss function is also simpler and less computationally intensive. The results are also good enough to work on the basic cases that this work is restricted to and so the swap

is made.

Countless hours can be spent looking for changes to increase performance. More have been made in this work, but these have been the most significant. The focus of this work will shift to improve the accuracy of the system, allowing it to work on more complex cases.

## 3.3 Summary

This chapter explained the hardware and software architecture of this work. The hardware defines an inexpensive foundation for real world testing and the software can be adopted to more advance hardware in the future. The software's modularity allows for additional information, from new sensors or forms of processing, to be added and considered in the behavior estimation block. This modularity also allows for quick iteration with software changes as each section can be encapsulated and ran separately.

Chapter 4 will present the main contributions and how all the reviewed matrial fits into the system.

# Chapter 4

# Distracted Driver Detection and Methodology

## 4.1 Processing Overview

The processing pipeline follows the data from the input of video to the output of behavior inference and vehicle location. The diagram in figure 4.1 highlights the steps involved. First, the video passes though the image processing block, which detects vehicles and spawns' trackers. The output are bounding box locations of vehicles of interest. Next, features are extracted from the bounding boxes and sent into both the anomaly scoring and machine learning block. The anomaly scoring block assigns, per frame and per track, a score composed of a peak detector and an oscillation detector. The output of this block is a score value representing how anomalous the signals current state is. The machine learning block will cluster, per frame and per track, the features movement as a sub-feature decomposed time series signal. The output of this block is the current cluster assignment, and that clusters average score. The output of the previous two blocks is then feed into the behavior estimator which infers the behavior of the vehicle and outputs, per track, the inferred behavior. The behavior is then used to annotate the video where the bounding box is drawn around the vehicle and its color corresponds to its behavior.

## 4.2 Image Processing

The details of the image processing block are shown in figure 4.2. Its objective is to take in an image and output the features for each tracked vehicle in the frame. The frame first gets downscaled and converted to a tensor for input into the convolutional neural network-based classifier and single shot detector. The output of this is a bounding box, a classification label, and the confidence level for each object. These detections are then filtered by label, car and truck are the only things needed to track, and the detectors confidence level in recognizing the object. It tends to output many low confidence detections. The detections are then ordered by which one is closest to the center of the screen and tracks are created in that order.

When no tracks exist, the first time a detection comes into the system one is created. After that, the tracks and detections are combined in the tracker association block. The typical reason for this association is to save computation time as the detector usually takes more time to run than the tracker. Here, instead, the tracker is used to supplement the detection difficulties for occlusions commonly encountered in the self-driving vehicle space. These are typically caused by glare, rain, clutter, shadows and various other obstructions.

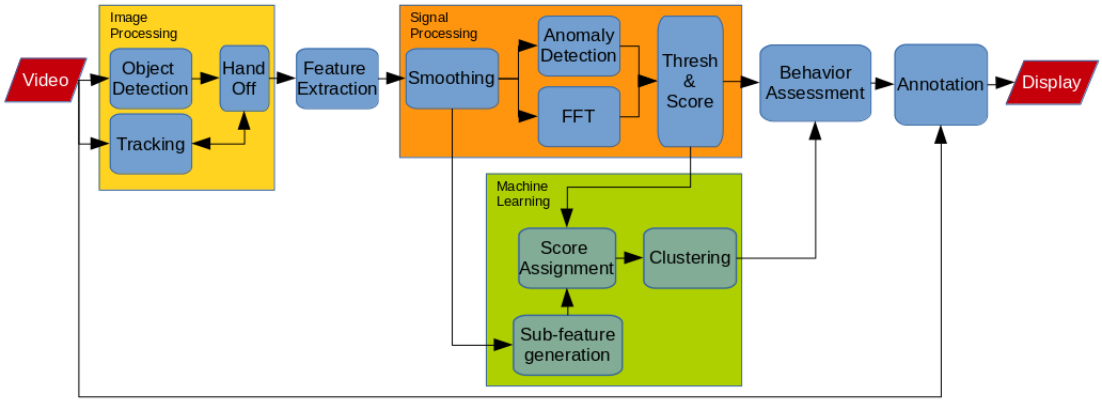



Figure 4.1: This is the processing pipeline which highlights the data flow though the entire system. It starts from the input of a video file or stream and ends with an annotated video file or stream.

This pairing allows for a more robust track through each frame. It is important to keep a continuous representation of the vehicle's movement. To find the relation between an existing track and a detection, a separation metric is calculated by the Euclidian distance between the incoming detection and an already existing track [70].

A pair is made between each new detection and existing track by determining the minimum distance of all detections against all the tracks. When this minimum distance is within a set threshold the detection is associated with the track. This means that the tracker has no need to run as the object's location can be updated with the detector alone. When a track fails to get an associated detector, the tracker is executed to find the object. When that fails, the track is removed. If a detection fails to associate with a track a new track is created. The most important reason for this is to create a unique identifier (UID) for each object. The UID allows the system to correctly relate the behavior of a vehicle between frames.

Once the final tracks are decided the resulting bounding boxes are converted into a time series set of features comprised of the object's width, height position in the frame and its area.

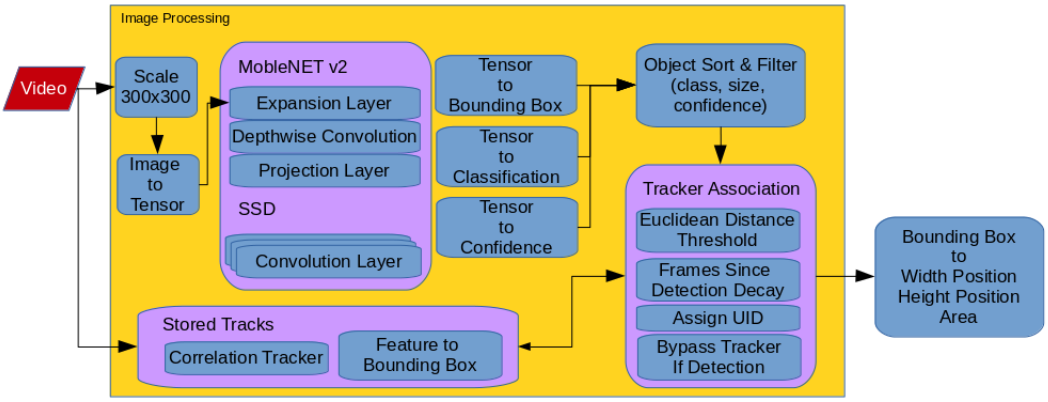



Figure 4.2: This block is the most computationally intensive and has a complicated pipeline which the entire system depends upon for proper operation. It starts with the video input and ends with the bounding boxes of all tracked vehicles.

Table 4.1: The table describes the parameters involved in the image processing block.

| **Parameter** | **Description** |
|---|---|
| Minimum detection confidence | When a detection is returned by MobileNet SSD it has a confidence value from 0 to 1 and this threshold sets the minimum confidence allowed into the system. |
| Maximum detection amount | The MobileNET SSD allows a limit on the returned detection amount where the detections returned are sorted by confidence and this threshold sets that limit. |
| Maximum bounding box size | To further filter detections, only realistic sized vehicles are passed on (It often mistook bridges for trains), and this sets the max area for the bounding box. |
| Minimum confidence to track | The detections above the previous minimum can be displayed, but a more aggressive threshold prevents them from being tracked and used for behavior inference. |
| Maximum amount of tracks | The maximum about of vehicles to track is set due to performance constraints. |
| Maximum distance for object association | This sets the max allowable Euclidian distance between an incoming detection and a track for an association between the two to be made. |
| Frame amount to forget track | This sets the number of frames (failed correlations or failed detections) allowed before a track is dropped completely. |
| Minimum track correlation | The correlation tracker returns how well it thinks its result correlates with its feature vector and if it's below a certain amount that track is considered a fail. |

## 4.3 Signal Processing

Figure 4.3 looks to take in features for each track and assign a score as to how anomalous that features movement is at the current frame. The scoring is done in two parts. First is the anomaly detector which, for each feature, flags the signals movement in the current frame as anomalous or not. The features that get flagged are the smoothed width position in the frame, the derivative of the tracks area and the derivative of its height position in the frame. Each of these features are weighted differently and assigned a score based on, through observed test runs, how representative each one is of the vehicle's movement.

Second is the oscillation detector which uses a moving window over the features to take an FFT. The bins of the FFT relating to low frequency components are used to create a score for that feature's oscillation. The score from the oscillation detector is applied to the entire moving window. The final output is a total score for each track, per frame.

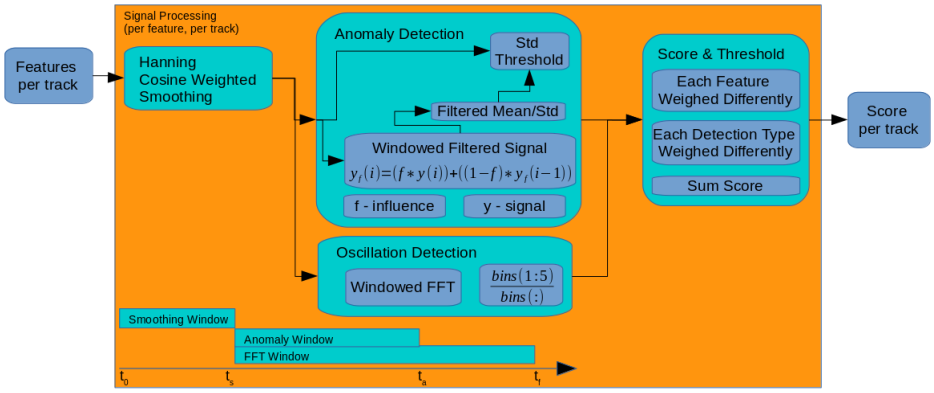


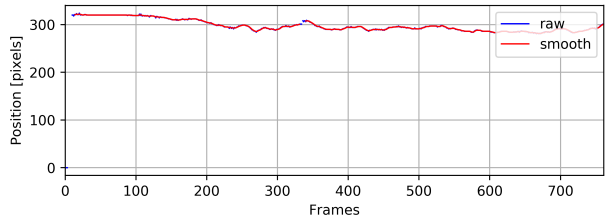

Figure 4.3: The signal processing block takes in the output from the image processing block and assesses the vehicles movement and give it a score based on how anomalous it is.

## 4.4 Machine Learning

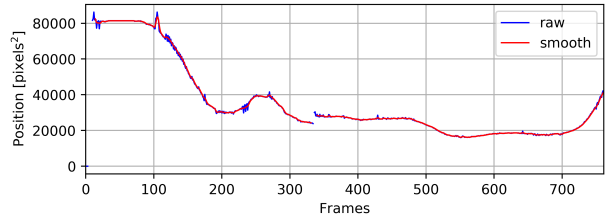
The objective of this block, figreu 4.5, is to group and label similar signal movement, per track and per frame, such that when anomalous movement is detected it can be related to a specific group of related signals. A moving window is used to calculate a set of sub features

Table 4.2: The table describes the parameters involved in the signal processing block.

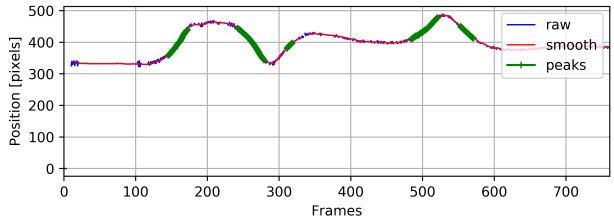
| **Parameter** | **Description** |
|---|---|
| Zscore - threshold | This parameter sets the standard deviation amount from the mean for a signal to be flagged as anomalous. (See description in Chapter 2 for more detail) |
| Zscore - lag | The lag sets the moving window sample amount for which the mean and standard deviation are calculated over. (See description in Chapter 2 for more detail) |
| Zscore - influence | The influence is a percentage of how much the current signal value effects the detection threshold. (See description in Chapter 2 for more detail) |
| Anomaly minimum frame time | When a signal is flagged as anomalous it must remain flagged for this minimum frame time before it can affect the score. |
| FFT - window | This is the sample amount used for the FFTs moving window. |
| FFT - bins | This sets the amount of frequency bins that the signals energy is divided between. |
| Feature score weights | Each feature that is used in the anomaly detection and FFT calculation is assigned a weight which is used to calculate its score. |



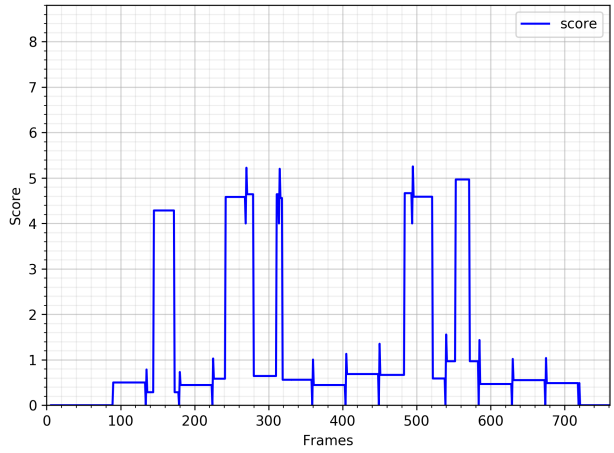

(a) Height Feature Example




(b) Area Feature Example




(c) Width Feature Example




(d) Score Example

Figure 4.4: The output of the signal processing block is the final score image for each track. The preceding images are an example of each of the features.

for each signal, per frame. These sub features are representative of the signal movement during the window and include statistics such as the longest period of time where the signal values are above the mean, total power in the lowest five bins of an FFT, mean error from a 3-sample mean forecaster, and the first minimum of an autocorrelation. The idea is that these sub features will better represent the movements of the signal that highlight the behaviors of a distracted driver.

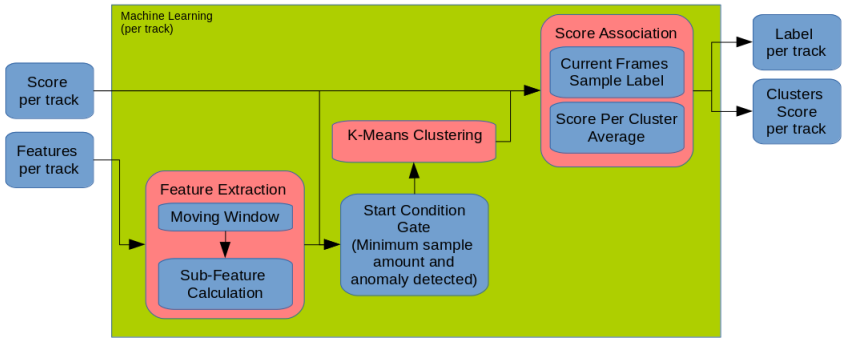


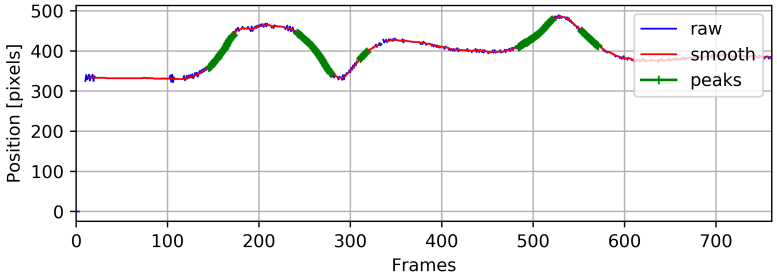

Figure 4.5: The machine learning block illustrates its unsupervised nature and it allows for learning from its environment in real time.

Next, these sub features are sent into a clustering algorithm, K-Means Mini Batch, for grouping. At each frame-step a new sample is added to the clustered set, the centers are recalculated, and the new sample is assigned a cluster label. Once assigned, the average score per cluster is calculated, where the score for each sample is averaged for each cluster. This provides a hint that the highest scoring cluster are likely associated with anomalous movement.

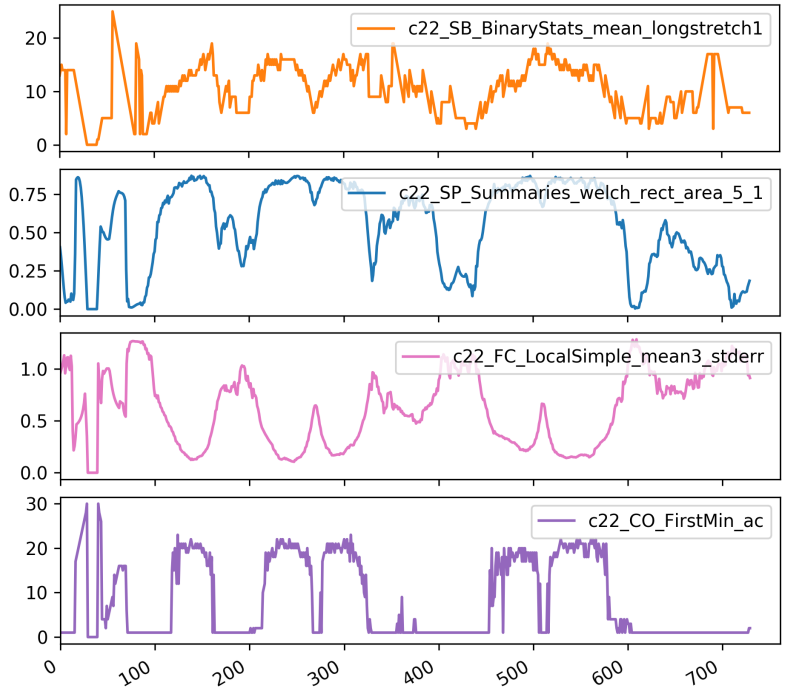
The final outputs of the block are the tracks cluster label and the average scores of all the clusters.

## 4.5 Behavior Estimation

The behavior estimator is the final step in the decision chain. It takes in the output from the signal processing and machine learning blocks. This includes each tracks anomaly score, cluster label, and cluster scores. It then makes an estimate about the behavior of

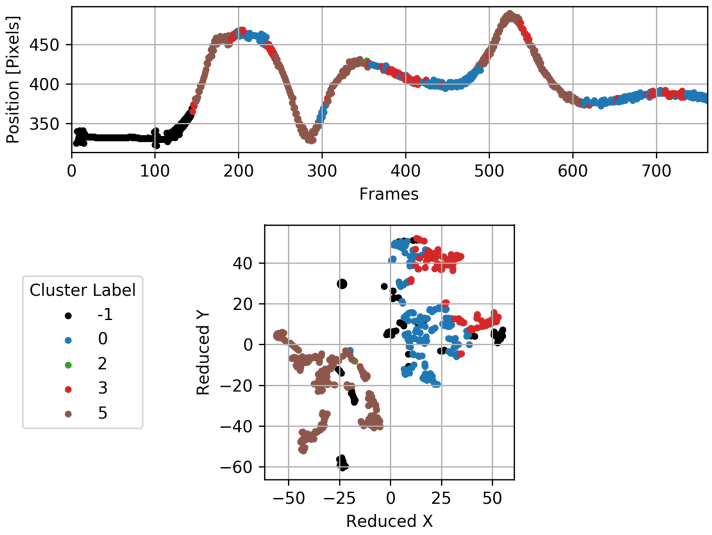



(a) Width Feature Example

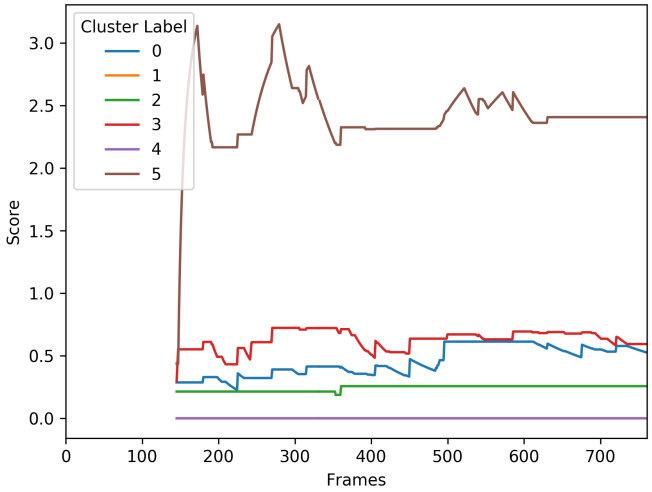



(b) Sub-Feature Extraction Example

Figure 4.6: The example above shows how the sub-features relate to the original signal. The sub-features calculated are the longest period of time where the signal values are above the mean, total power in the lowest five bins of an FFT, mean error from a 3-sample mean forecaster, and the first minimum of an autocorrelation




(a) t-SNE Cluster Example




(b) Cluster Score Example

Figure 4.7: The score example plot shows the cluster score with time as it relates to the clustered signal above. It shows how the score is averaged into the cluster for each sample and that the cluster related to the sharp signal movements is associated with the highest score.

Table 4.3: The table describes the parameters involved in the machine learning block.

| **Parameter** | **Description** |
|---|---|
| Maximum cluster Size | The cluster will reset after the sample amount reaches this maximum value. |
| Minimum cluster Size | Before clustering occurs this minimum sample about must be reached thus allowing the clusters to form around a batch of samples before adding them individually. |
| Cluster label amount | The number of clusters to create from the given sample space. |
| Sub-feature window size | This is the moving window used to calculate each of the sub-features. |
| Sub-features | The set of sub-features used. |

the vehicle being tracked. This estimate is split into three parts, with each part being its own state in a state machine, normal is when a vehicle is driving as expected, abnormal is when its position is deviating from the intended path, distracted is when it continues to deviate, and evidence from the vehicle's movement history shows a strong irregularity. A few expert-defined conditionals have been created to decide when to transition from one state to another.

The conditionals set to transition from normal to abnormal are based primarily on the anomaly score and partly on the current cluster and the history of its score. For the anomaly score, it must be above a set threshold for a certain number of frames for the transition to be made. The scoring function and threshold are tuned such that it will only be exceeded with an obvious change in vehicle movement. The change could be sudden, a sharp swerve, or slowly driving out of a lane. The cluster-based conditional will, if met, transition with a set probability as the conditional is not always a sign of abnormal behavior. It is met when the track is in a cluster for a certain number of frames and its score is above a threshold. This implies that its current cluster label is assigned to a cluster that was previously known

to be associated with erratic behavior.

The transition from abnormal to distracted occurs only if its current cluster assignments score is above a threshold and is among the highest scoring clusters. This would allow the clusters score to accumulate over repeated movement such that a revisit to a cluster known to be related to distracted driving would trigger this transition quickly. If it fails to transition to distracted after a window of time and the anomaly score is no longer above a set threshold, then it de-escalates back to normal.

When in the distracted state, conditionals are set to transition to normal or to override the de-escalation and keep it in the distracted state. To transition back to normal, the anomaly score must be below a threshold for a window of time, the current clusters score falls below a threshold, or if the clusters label changes into a low scoring cluster after a window of time. The keep conditional will maintain its state if the label has not changed for a certain number of frames. This is to provide the state transitions with a sticky property such that an amount of time needs to be waited after a transition before making another decision. The final output is the behavior estimate, per track, as Normal, Abnormal or

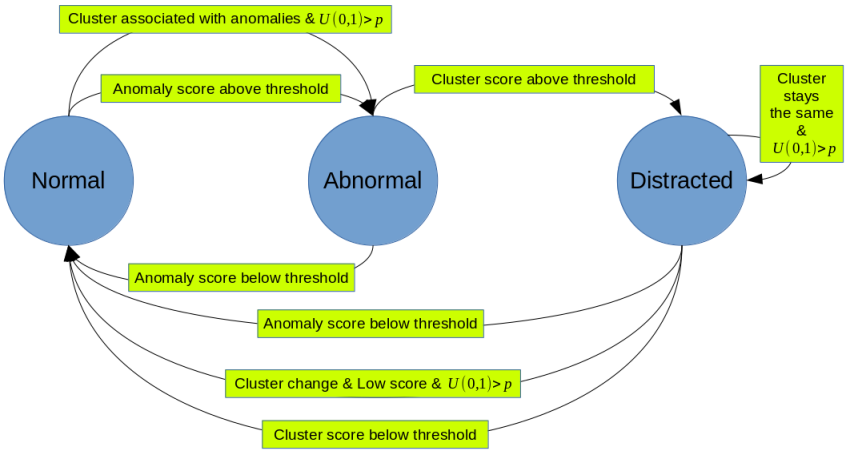



Figure 4.8: The state diagram shows the simplified transition conditionals between states and highlights its ability to use lesser likely conditionals to tradeoff between missed maneuvers and false positives. The three behavior outputs of the system are Normal, Abnormal and Distracted.

Table 4.4: The table describes the parameters involved in the behavior estimation block.

| **Parameter** | **Description** |
|---|---|
| Minimum state transition time | To prevent jumping around states too quickly each transition combination has a minimum amount of time required before another transition may occur, specified in frames. |
| Anomaly moving window matrix | The moving window matrix specifies the past window of time, in frames, that each conditional use to make its decision. For anomaly this is the windowed score. |
| Anomaly score threshold | A fixed value that the score must be higher than before any decisions are made. |
| Anomaly conditional probability matrix | This is a matrix of probabilities which define how likely a state transition is for each anomaly-based conditional. Set from 0 to 1. |
| Clustering moving window matrix | Same as above for all clustering related measurements. |
| Clustering minimum score | A fixed value that the clusters score must be higher than before any decisions are made. |
| Cluster conditional probability matrix | Same as above for all cluster-based conditionals. Set from 0 to 1. |

Distracted.

## 4.6 Base Assumptions and Driving Scenarios

The base assumptions for this work are that the vehicle the camera is mounted on, the tail car, is driving straight and the vehicle being tracked, the lead car, is attempting to drive straight. This is the case most of the time when traveling on a highway and for stretches of a main road. These assumptions allow the work to focus on tuning and building the system to first handle a basic case while allowing for expansion later.

Scenarios can be crafted to still thwart these basic assumptions, which lead to false positives, such as the existence of a pothole in the road. The lead car swerves to avoid it such that the entire car is on the left side of the pothole. The tail car allows the pothole to pass in between the tires, putting it beneath the car, and it would maintain a straight path. This would cause a false positive as the lead car is not distracted and is a great example to highlight the current limitations of this work. The behavior designation is inferred through the lead vehicles movement.

To verify the system's ability to function under the base assumptions, six datasets have been gathered from various sources. These sources include driving and recording on the systems hardware, simulation using CARLA [71], and videos from YouTube [72]. Four of the datasets, each containing 20 videos, aim to test two specific maneuvers that are commonly caused by distracted driving. For each of these maneuvers, a set of videos from hardware and from simulation are used. The first maneuver is caused when a driver is attempting a task on a cell phone, which takes their attention off the road. It is best described as a 'drift,' where the vehicle begins to slowly deviate off its intended path followed by a correction back to the path. The second maneuver would be best described as a 'swerve,' where the driver quickly changes direction and then back again. This would happen should the driver fumble thier phone or drop a sandwich. In each of these videos only one maneuver is preformed, but with variability in the strength and duration. Variability also comes from car type and for the simulated videos, location.

The last two datasets are a mixed bag of various driving behaviors as seen in a parking lot or on the road. Some videos contain many maneuvers from the same vehicle and of various types. Others contain nothing at all or subtle movements. The goal of this dataset is to test the system in a variety of scenarios. Each video in the mixed sets has a naming convention to help outline the goals of each video. The naming convention for the videos used is:

DX_CY_Date_Location_Vehicle Of Interest_RVS(if camera is facing backwards)
Where Y = The level of clarity of the vehicle: 0 being perfect 5 being difficult
Where X = The level of distraction from 0 to 5 as described in the table below

Table 4.5: The table describes the different actions a driver could take to cause distracted maneuvers

| **Severity** | **Title** | **Impairment Type** | **Description** | **Observables** |
|---|---|---|---|---|
| D1 | Quick Check | Visual | A moment of averted gaze, less than 1 second, looking away, but not engaging with other devices | Slight swerve or delay |
| D2 | Task Attempt | Visual, Manual | Frequent looking away, in bursts of 0-1 second, engaging device (changing song, responding to text) | Slight drifting |
| D3 | Task Fixation | Visual, Manual, Cognitive | A full 1-5 seconds of not looking (responding to a text, confused by GPS) | A slow drift and a correction |
| D4 | Distracted | Visual, Manual, Cognitive | Combination of D2,3, but persisting over long periods (watching a movie, reading a book) | Continuous drifting, missing lights, no blinker |
| D5 | Erratic | Visual, Manual, Cognitive, Emotional | Drunk driving, Angry driving | Obvious swerving and dangerous movement |

Figure 4.9: Screenshots from video file D0_C3_4-21-2019_84to90E_None. Nothing of significantes occurs in this video. Nothing of significates occurs in this video. It helps to highlight the robust nature of the system by testing it when vehicles are behaving normally.

An extensive list of videos may be found in the appendix. To provide a few examples, in the video D0_C3_4-21-2019_84to90E_None no distracted driving occurs, but plenty of vehicle movement happens around turns and with lane changes. D3_C2_6-27-2019_WPIpark_BrSW is a video, where the driver is acting as someone who is fixated on a task causing them to drift and sharply correct. D5_C1_5-11-2019_BHS_CCtruck is a video, where the driver purposely swerves around an empty parking lot.

Every video in the datasets have been labeled with a zero when the vehicle is driving normally and a one otherwise. Specifically, when the vehicle appears to be distracted as subjectively decided by a human labeler. The labels look to reduce the problem to a binary hypothesis testing problem where, at each frame, the system must output a one or a zero.

## 4.7 Summary

This chapter explained the methodology used to connect all the pieces together and the reasoning for doing so. Many of the solutions presented came about through testing and observation of the datasets. Each dataset has a purpose in performance evaluation from robustness to simple maneuver detection and each provided insight to the systems design.

Chapter 5 will present the results, explore parameters, and application specific usage.

Figure 4.10: Screenshots from video file D3_C2_6-27-2019_WPIpark_BrSW. This video shows a driver imitating a distracted driving maneuver known as a drift.

Figure 4.11: Screenshots from video file D5_C1_5-11-2019_BHS_CCtruck. This video shows a truck driving around a parking lot in an erratic manner.

# Chapter 5

# Experimental Results

The experimental results which follow aim to highlight the performance of the system under the base assumptions where both lead and follow vehicles are attempting to drive straight. Standard binary hypothesis testing metrics are first summarized. Next, application specific metrics are introduced, which highlight the systems performance when used in an example application. The systems parameters are then explored, and key ones are highlighted which effect performance the most. This helps to give insight on how the system can be adjusted for certain needs.

## 5.1 Metrics

The proposed system will be assessed with a few different metrics which have been chosen to measure performance in different areas. The end comparison between the system output and truth will be a binary hypothesis testing problem, thus leading to a variety of existing metrics to choose from. The goal of this system is to correctly identify a distracted driver, given the base assumptions in Chapter 4. It is also important to be robust and minimize the number of false positives. The metrics are chosen to highlight the relation and show the tradeoffs made between the two.

Accuracy is the most common, but often fails to highlight performance in testing sets with a large class imbalance [73]. It may not seem obvious at first, but the system makes a call on a frame by frame basis. With typical driving situations a vehicle will be operating

normally most of the time. This leads to a much larger frame count for the normal class and thus more examples of a 0 than a 1. It is also important to note that decisions made at each frame are temporal and rely on past information.

To give insight to the shortcomings of the accuracy metric two other numerical metrics will be provided. The Matthews Correlation Coefficient (MCC) was designed to combat class imbalance and is one of the better metrics to use when assessing binary classifiers [74]. The F beta score is used to allow an adjustable importance of precision verses recall [75]. Where precision is the ratio of true positives to all true guesses and recall is the ratio of true positives to all positives. It is important to realize that the F score does not consider the number of true negatives.

The final numerical metric used is one which focuses on the application of flagging a distracted driver and does not penalize on the timeliness of its detection. This metric is called the hit and miss count. A hit is when the prediction is correct anywhere within the time window of the maneuver, multiple correct predictions within the window are only counted once. A miss is any incorrect prediction outside of this window. When using them over a set of videos the hit rate will be the ratio of the hit count to all maneuvers that have occurred (true positives). The miss average will be the average of the miss count though all videos. This will only be used to assess the dataset, which all contain only one maneuver.

For each video, the confusion matrix will be shown along with a plot showing the frame by frame decisions made by the system compared against the truth labels. In addition, each video will be plotted in the receiver operating characteristics (ROC) space so that the trade of between the true positive rate and the false positive rate is obvious [73]. Specifics for each of the metrics follows:

Accuracy is measured from 0 to 1, where 1 shows the best performance and 0 is the worst (although as noted above the values in-between are not as telling). It is calculated from [73]:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{5.1}$$

where $TP$ is the true positive count, $FP$ is the false positive count, $TN$ is the true negative, and $FN$ is the false negative count.

F beta score is measured from 0 to 1, where 1 shows the best performance and 0 is the worst. It is calculated by [75]:

$$F_\beta = \frac{(1+\beta^2) * TP}{((1+\beta^2) * TP) + (\beta^2 * FN) + FP}, \tag{5.2}$$

where $\beta$ is the factor which selects the imporance of the recall where the higher the $\beta$ the more valued the recall. For this work $\beta = 2$.

The MCC is a value from 1 to 1, where 1 is when the predictions are completely reversed, 0 is random guessing and 1 is a perfect prediction. It is calculated by [74]:

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}, \tag{5.3}$$

The miss average is the mean of all misses that have occurred in the video and is represented as a positive value from 0 to the sample size of the largest run. The best value is a 0:

$$MissCount = \sum ContinuousFP \tag{5.4}$$

$$MissAverage = \frac{\sum MissCount}{VideoCount}, \tag{5.5}$$

where one $continuousFP$ is a set of false positive rising and falling edges.

The hit rate is the ratio between the amount of hits made by the predictor and the amount of hits in the truth data. The best value is 1 and the worst is 0:

$$HitCount = \sum_i \min(ContinuousTP \in P_i, 1) \tag{5.6}$$

$$HitRate = \frac{\sum HitCount}{TotalContinuousP}, \tag{5.7}$$

where $TotalContinuousP$ is the count of all manuvers in the video dataset and one $continuousTP$ is a set of true positive rising and falling edges

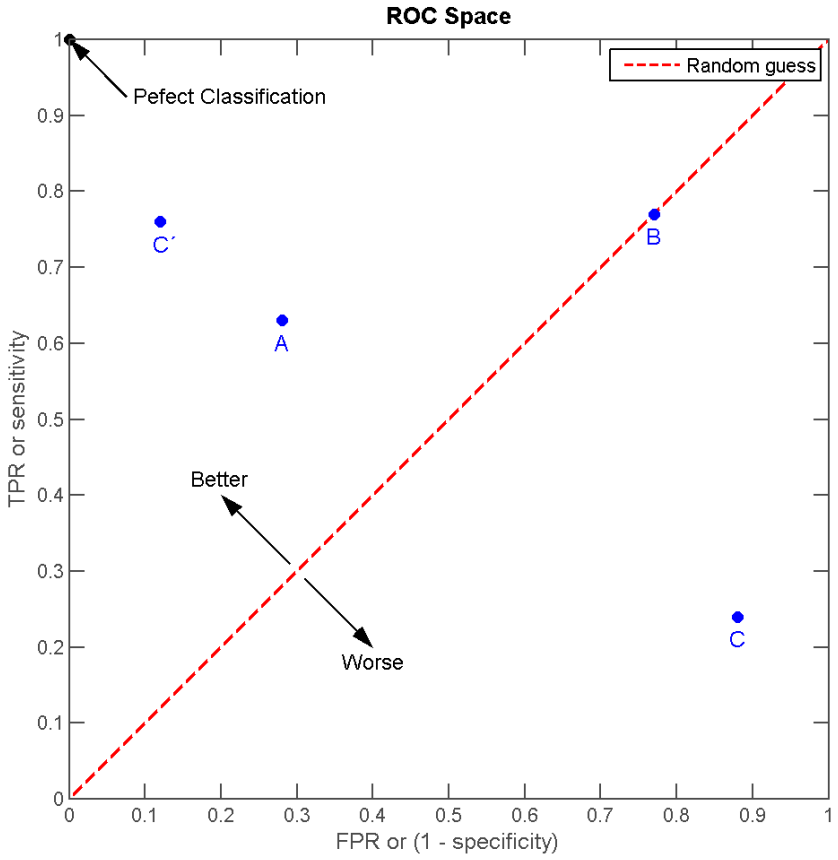



Figure 5.1: An example of an ideal point and the worst point are shown in the ROC space and, the diagonal dashed line represents random guessing [73, 76].

## 5.2 Adjustments

Initial results were taken from the mixed hardware dataset to highlight the performance gains of certain changes. This set has the most variety in driving and environment. The parameters chosen to get these results where decided on though observation and deduction during the development of the code. This section provides an overview of how tuning parameters trades off performance on certain metrics.

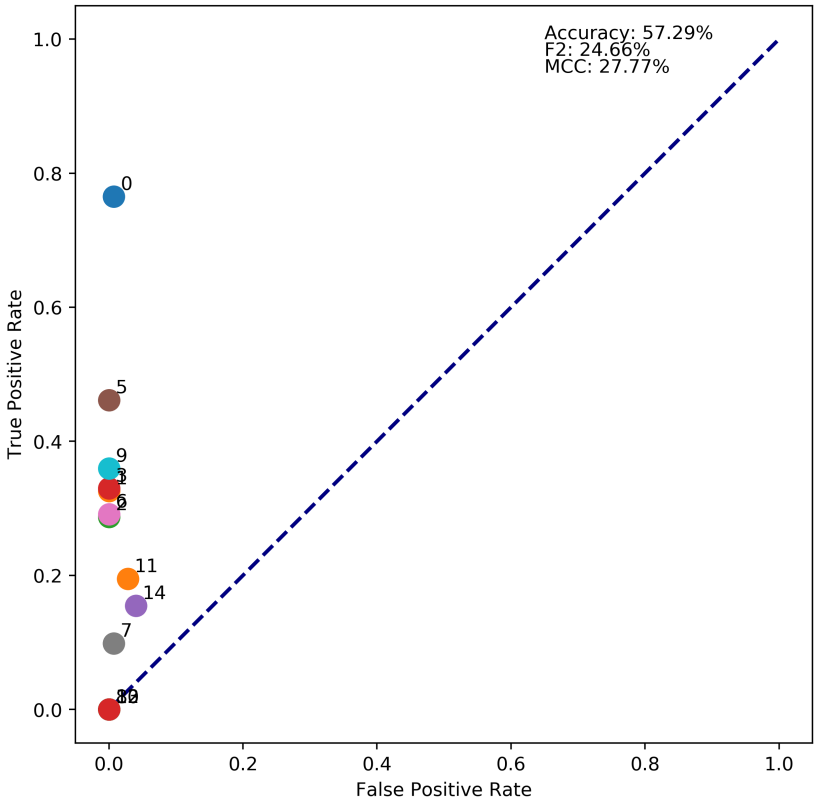



Figure 5.2: This receiver operating characteristics space shows the initial performance of the system after each block was optimized separately and put together. It shows from the dot locations that the system does its best to be robust to false positives in all videos.

The initial metrics are objectively not bad if minimizing the false positive rate is our goal. However, some critical maneuvers are missed completely. The tradeoff between missing them and getting more false positives is something that needs to be balanced. The images in figure 5.4 highlight the misses and why they are important.

After the initial tests it became apparent that additional conditionals where needed for the state transitions. However, the conditionals added dont hold true in all cases. This uncertainty is accounted for by using a stochastic state transition much like that used in a Markov decision process (MDP) [77]. Here the probabilities of those conditionals, as defined in Chapter 4, where adjusted.


Figure 5.3: The frame-by-frame and a filmstrip of snapshots from the video in question,D4_C5_8-31-2019_BHS_Blk4DS, show the behavior of the system in a simple case. In the video is one swerve maneuver.

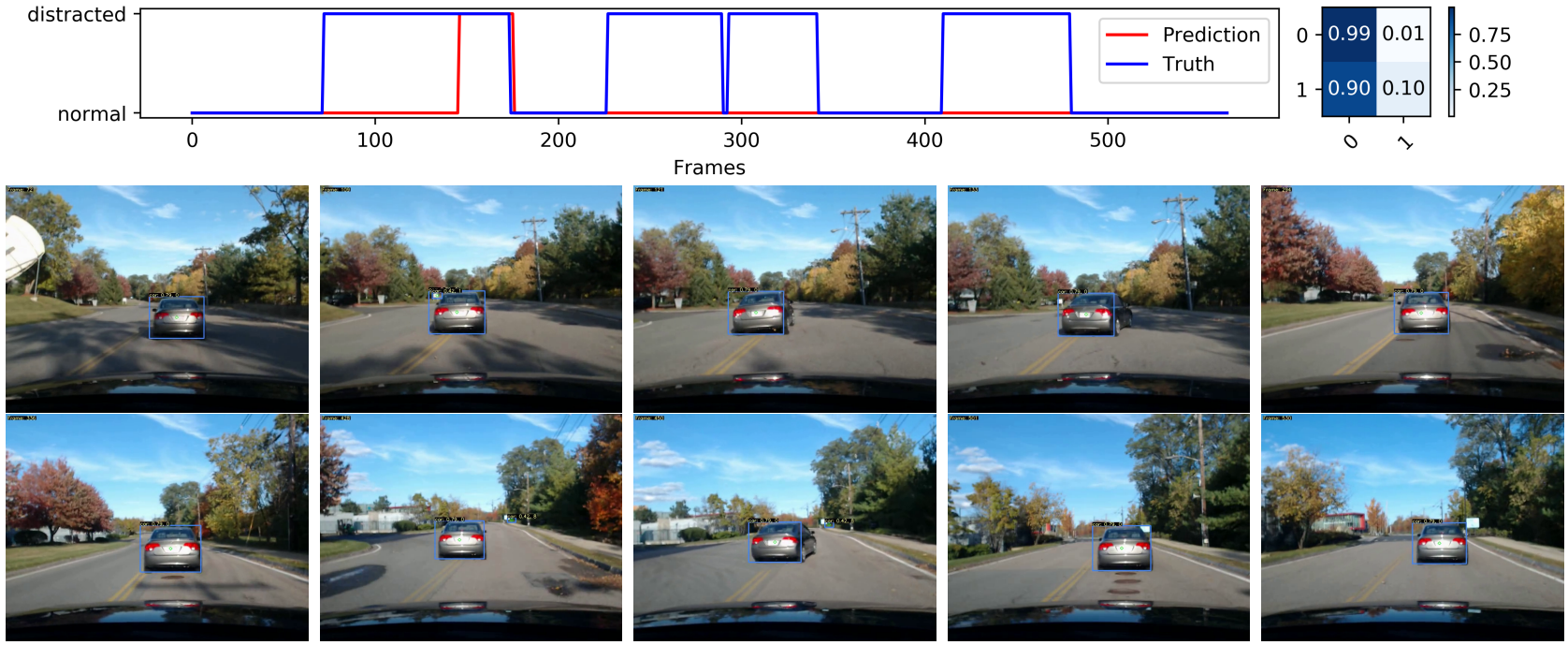
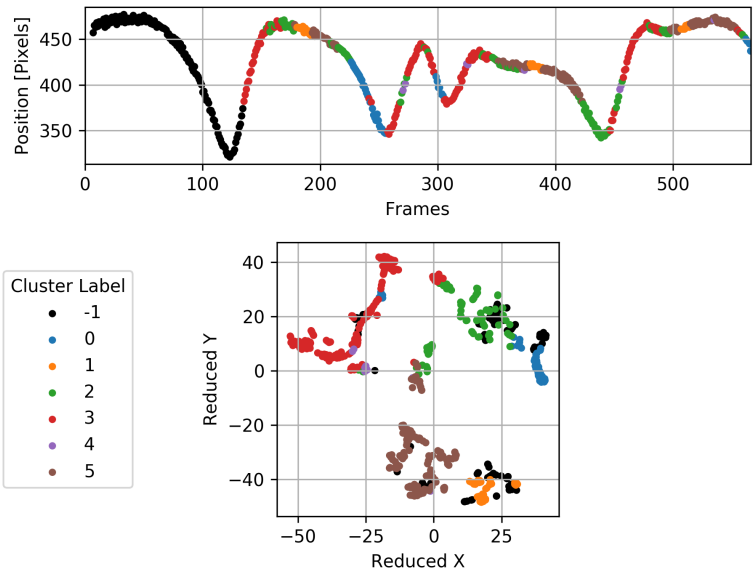
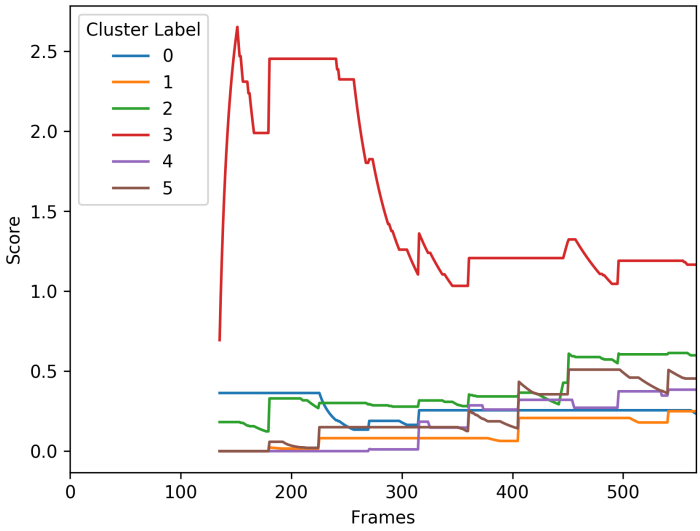



Figure 5.4: The frame-by-frame and a filmstrip of snapshots from the video in question, D3_C3_08-26-19_2nd_PrvRd_Civic, show the behavior of the system in a simple case. In the video are three swerve maneuvers.




(a) t-SNE plot of the vehicles width position




(b) Clusters average score

Figure 5.5: The t-SNE plot above shows the clustering performance and score of the previous run, D3_C3_08-26-19_2nd_PrvRd_Civic. These two graphs highlight that the system could recognize the distracted behavior despite the anomaly score falling below the threshold. The score for the cluster associated with the distracted behavior, cluster 3, is the highest scoring cluster of the group and stands out.

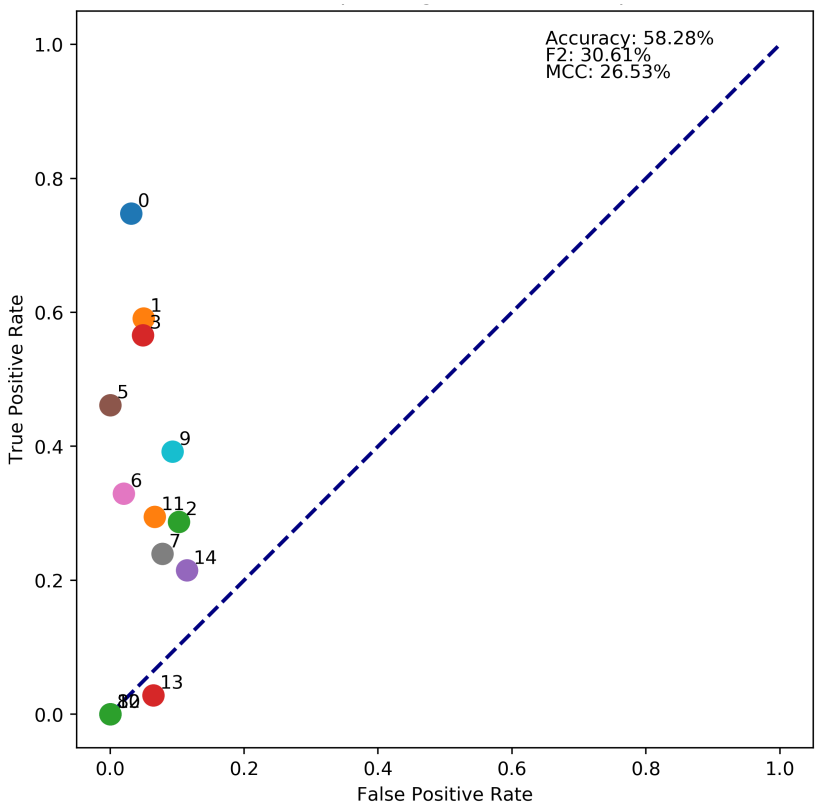



Figure 5.6: After the addition of new conditionals, the true positive rate for each video had mostly increased and brought the group of points closer together at a higher true positive rate while only adding a new false positives.

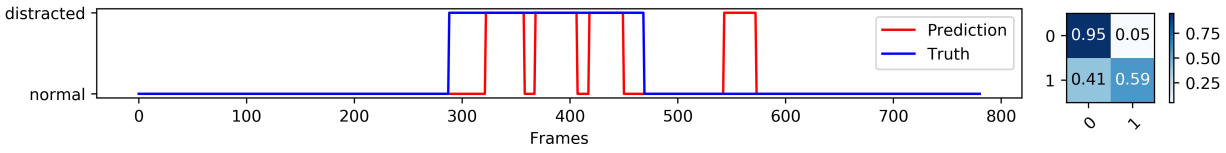



Figure 5.8: Shown is the same run from the initial results video, D4_C5_8-31-2019_BHS_Blk4DS, it shows addition false positives were created die to these changes.

The system contains many parameters as noted in Chapter 4. To fully explore the space and forgo the logic used initially to set them it would be useful to sweep them using sci-kit learns grid search CV technique [78]. The parameters that are difficult to set are


Figure 5.7: The frame-by-frame and a filmstrip of snapshots from the video in question, D3_C3_08-26-19_2nd_PrvRd_Civic, show the behavior of the system in a simple case. In the video are three swerve maneuvers.

the anomaly detection thresholds, the feature score weights and the behavior estimators' conditional probabilities. All of which are critical in the tradeoff between the miss rate and false positive tradeoff. Using the mixed hardware dataset these parameters where exhaustively searched to find the best set optimized my the MCC.

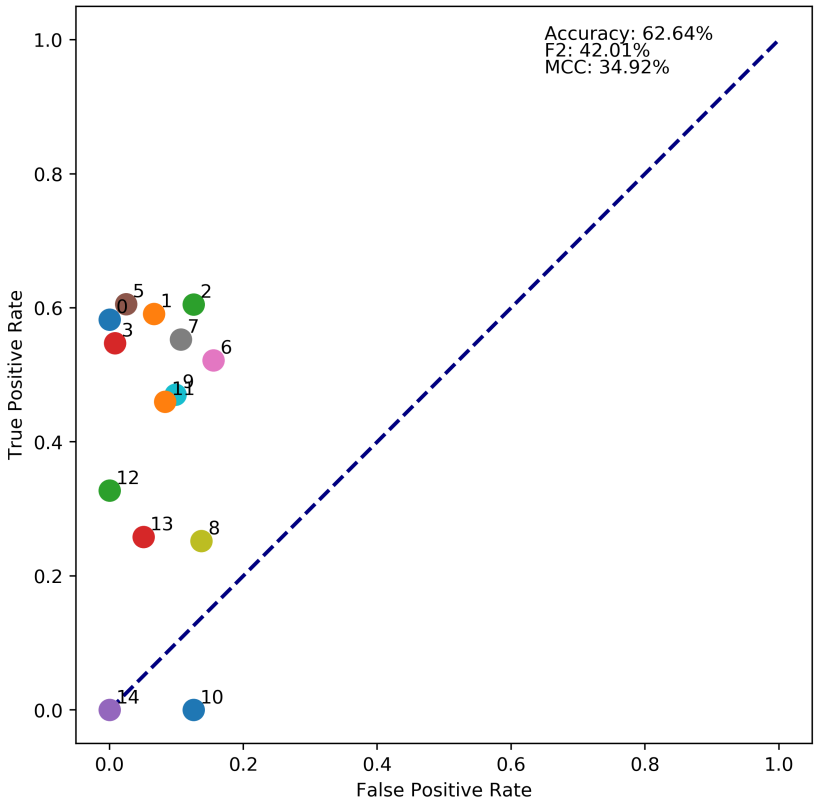



Figure 5.9: The parameter sweeping performance increase is debatably useful. The grouping for the videos had spread out and a larger number of false positives was let in.

As noted from figure 5.9, all our metrics have increased, and the cluster of points has shifted up and to the right signifying that compromises have been made in exchange for these better metrics.

In addition to a grid search, a one parameter at a time sweep was completed and plotted in the ROC space to generate a ROC curve. The parameters swept are the behavior estimators normal to abnormal cluster-based transition probability, the anomaly scores standard deviation threshold for peak detection and the window time used for cluster feature decom-

position. For simplicity, a subset of the videos was used. The four videos where chosen to represent the span of difficulties for the system: carla_drift_42_Town03_loc-0 is an easy drift maneuver, swerve_mini_6 is a difficult swerve maneuver, D3_C3_08-26-19_2nd_PrvRd_Civic is a mix of swerve maneuvers and D2_C2_6-27-2019_WPIpark_BrSW is a realistic drift maneuver. Each one with a different vehicle and in a different environment.

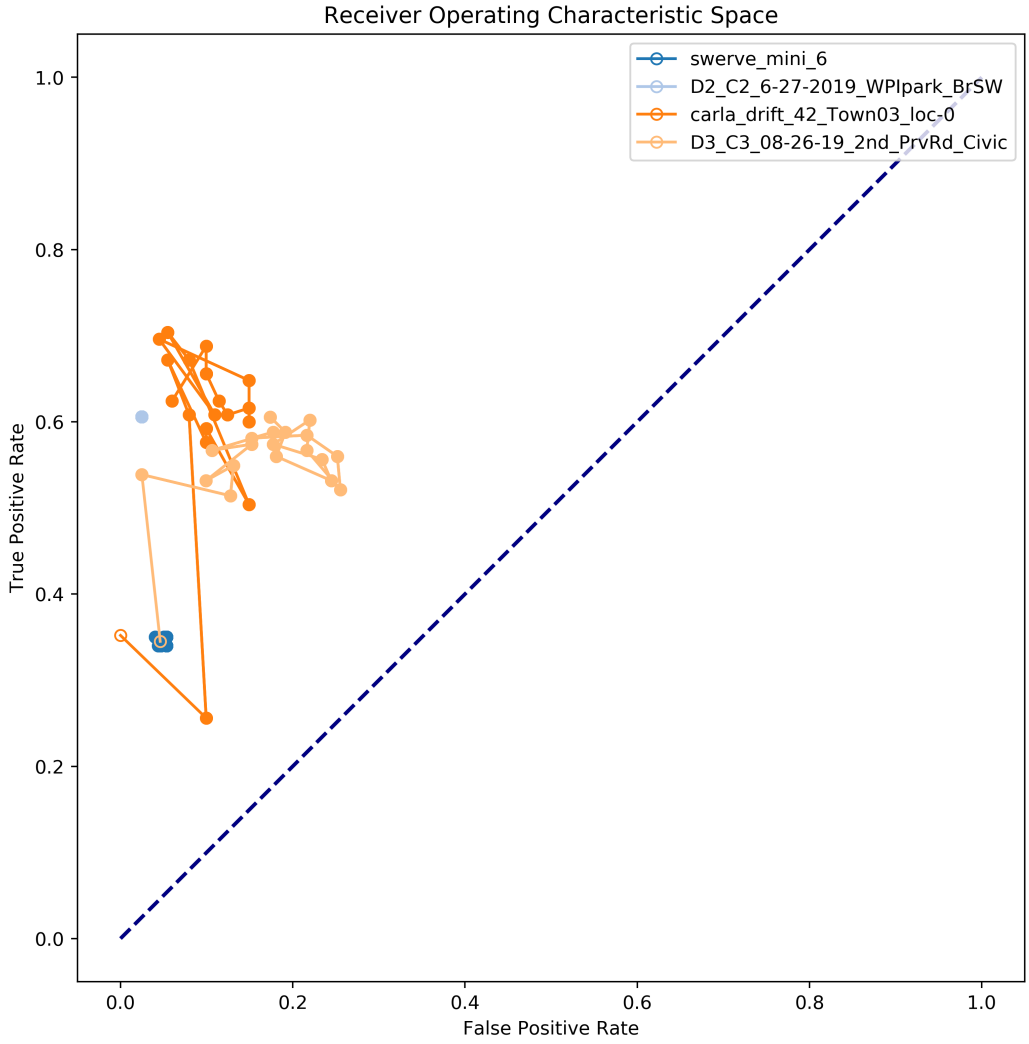



Figure 5.10: This ROC curve shows the parameter sweep of the behavior estimates transition probability from normal to abnormal. The sweep is 20 steps, linearly spaced from 0 to 1. As you can see from the plot, this parameter only effects the videos, which have multiple maneuvers as it helps to detect maneuvers which the anomaly score misses. As this parameter moves up from 0 (denoted by the empty circle) you can see the increase in the true positive rate and note this is a probability which is why the dots seem so sporadic.

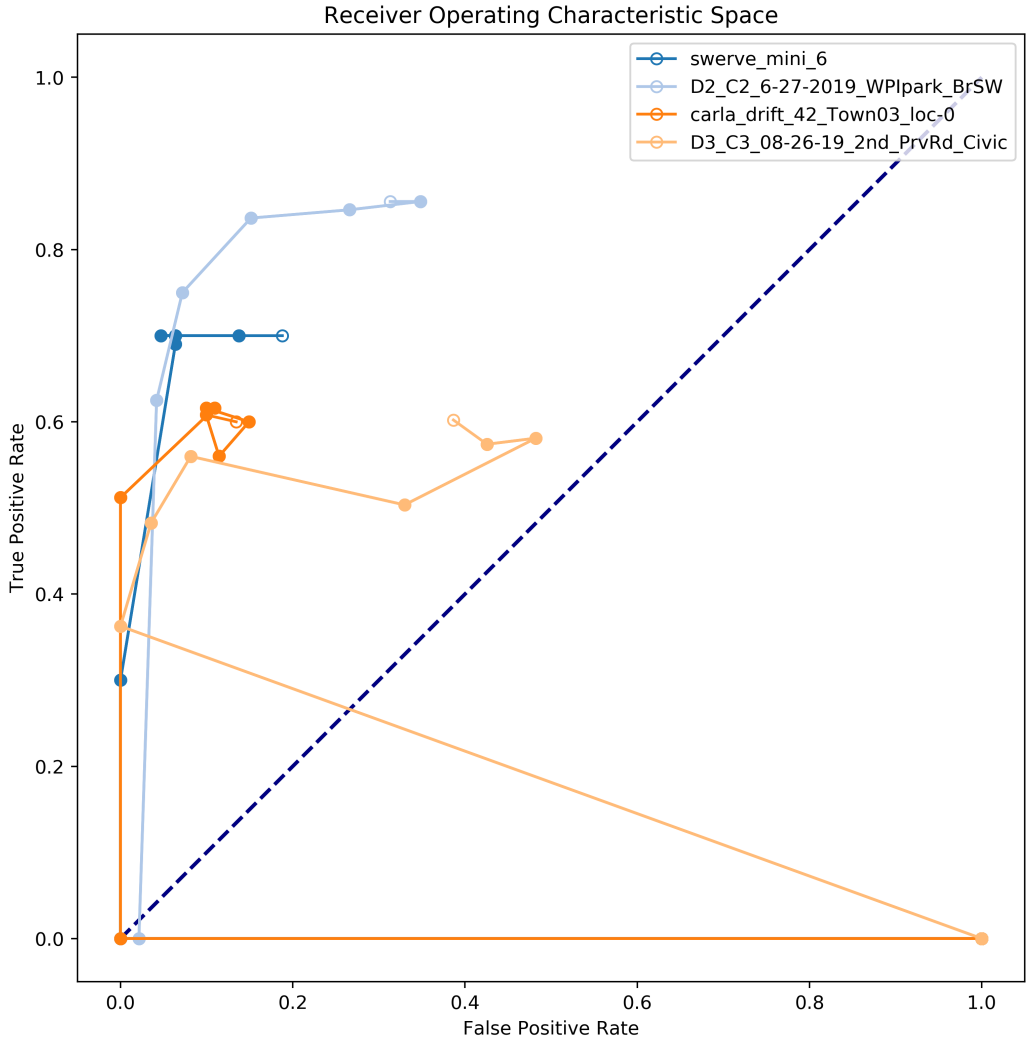



Figure 5.11: The ROC curve shows the parameter sweep of the anomaly score peak threshold. The sweep is 20 steps, linearly spaced from 0.5 to 5.0. The curve above highlights the importance of this parameter and its direct impact on the false positive rate. It starts at a low value (denoted by the empty circle) and as the value increases the false positive rate decreases as its more resilient to noise. Up until a point when nothing is detected.

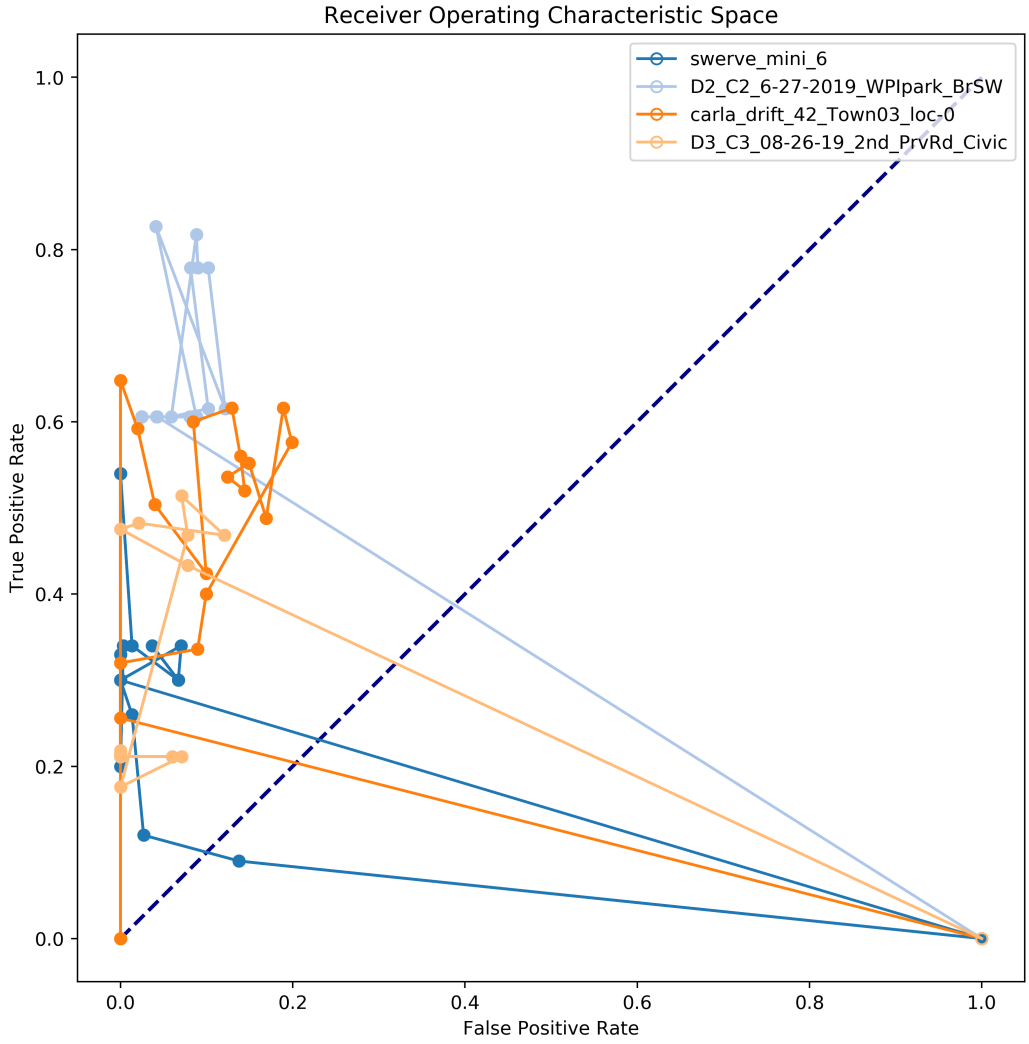



Figure 5.12: This ROC curve shows the parameter sweep of the window size used to create the features sub-features for clustering. The sweep is 20 steps, linearly spaced from 0 to 100. The value starting at zero (denoted by the empty circle) doesnt detect anything and as the value increases the features are better representative of the vehicle's movement and allows for better detections up until a point where is stops helping.

The ultimate choice of parameters is up to the application the system will be used for. This will be discussed further in the following section. In addition, the desktop PC used for this parameter sweep consistently ran out of memory and crashed causing the search

through the parameter space to not be complete. Going forward, for simplicity, the MCC metric performance will be favored.

## 5.3 Results

The system demonstrates its ability to recognize the drift maneuver in both datasets. The ROC graph gives a good overview of performance and in both datasets, it favors minimizing the false positive rate. The metric values and the overall true positive rate make it appear that the system is only recognizing half of the true positives. However, the frame-by-frame decision graph highlights the reason for these seemingly low values. Estimates are made for each frame and when the driver is engaging in a distracted maneuver the truth label covers the continuous block of frames for the maneuver as if a human would recognize it. While it is true that the timeliness of the system does not line up with what a human would choose, it is important for the system to detect it even if its late, as it does in all cases. This is noted as having any true positives as each video in this set only has one maneuver. In an application requiring a tag and not real time information, the driver is still flagged, and the system still accurately marks the driver's vehicle. This can be shown numerically with the miss averate and hit rate metrics as seen in the ROC plots.

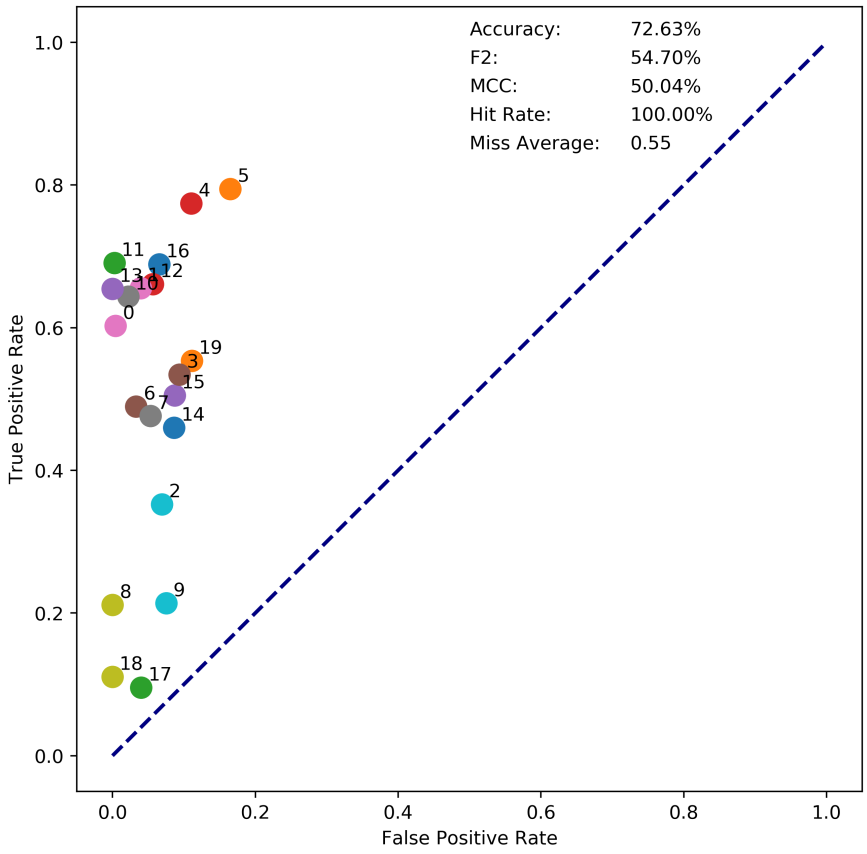



Figure 5.13: The ROC space for the hardware drift data set shows its ability to successfully mark every maneuver and have only a few false positives as the video dots are all to the left of the space. In some cases, notably videos 18 and 17, does the system only detect a small part of the maneuver. For the full performance figure see the Appendix A.

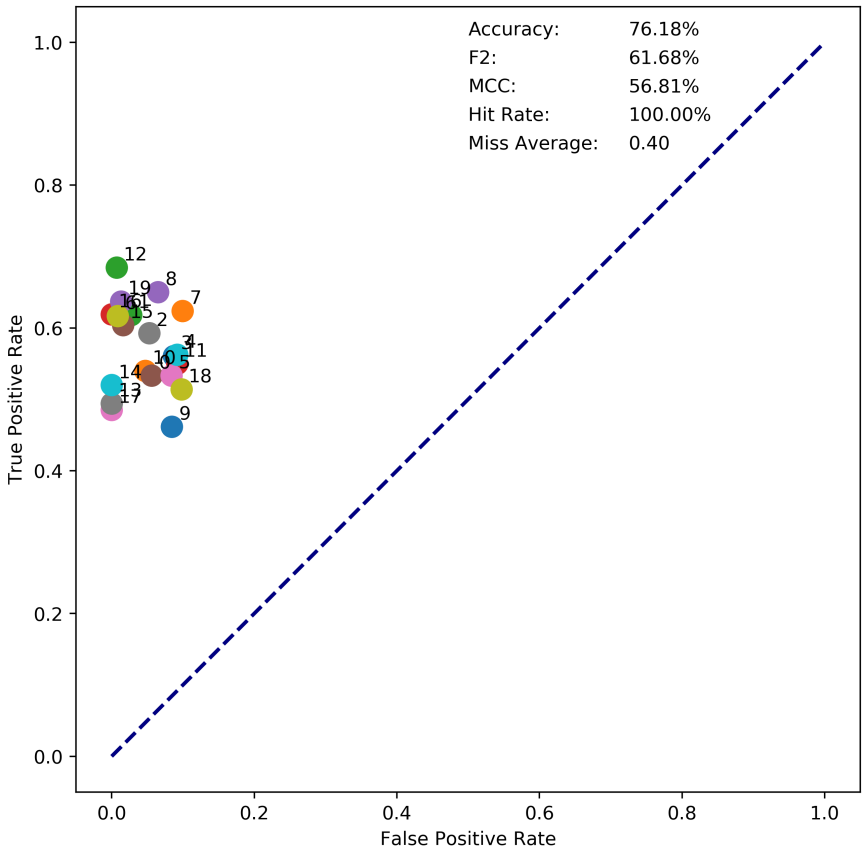



Figure 5.14: The performance for the CARLA drift dataset is like the hardware. Due to the vehicle's movement being generated though randomness programmed in a script the overall variance of movement is lower than having a human driver in the real world. This causes a tight grouping and great performance. For the full performance figure see the Appendix A.

The swerve maneuver is not detected as well as the drift maneuver, or at least it isnt on time. As you can see from the frame-by-frame breakdown it does detect that an irregularity is occurring, except it makes the assessment after the maneuver is completed. When this happens, the metric values dont reflect any positive performance. The swerve motion occurs quicker and more time is needed to asses it. This could be accounted for by adjusting the window times that are used to make the estimations as they are set such that at least a second of data is needed. This brings up another tradeoff of quicker response time verses more false positives.

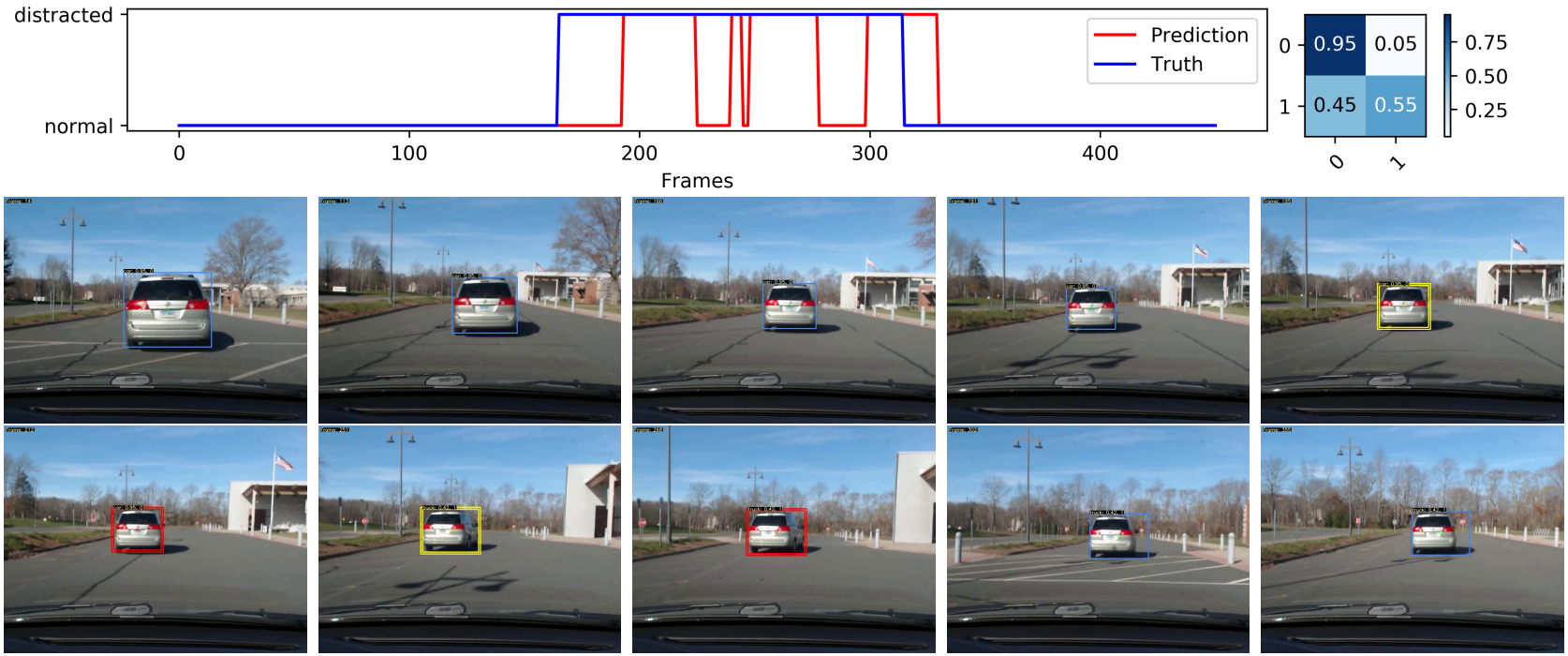



Figure 5.15: In this video the minivan is performing one drift maneuver. The filmstrip above shows the behavior estimate changing as the vehicle is in different parts of the maneuver as highlighted form the accompanying perdition and truth comparison plot.

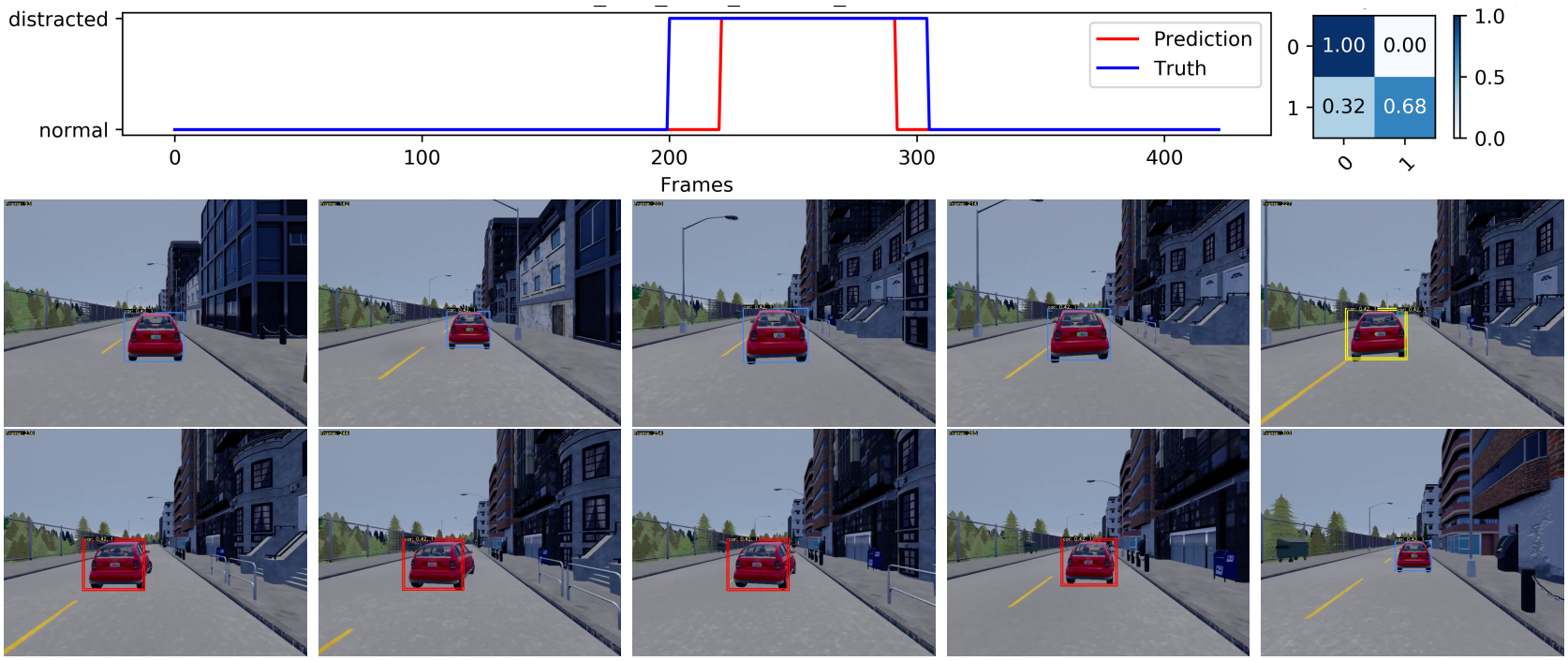



Figure 5.16: This is our first example from the simulator. It shows the vehicle driving from a scripted location and varying randomly the strength and duration of the drift. This maneuver is executed by the enabling and disabling of the simulators auto pilot feature.

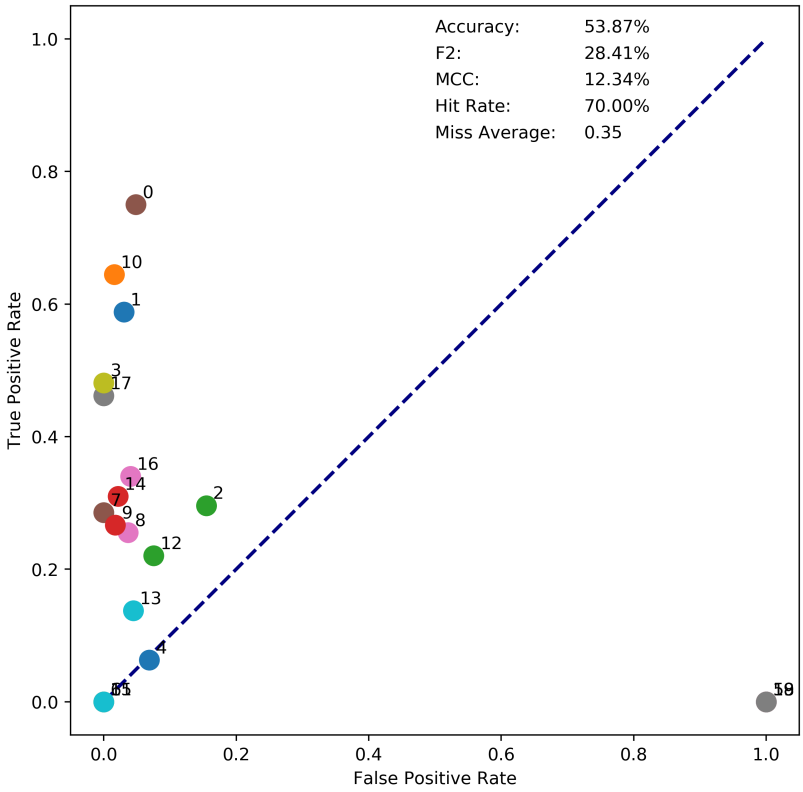



Figure 5.17: The ROC space for the hardware swerve dataset performs well and highlights an important issue with the way the parameters are set up. The serve maneuver is significantly quicker and although it does detect it 70% of the time it is typically late. For the full performance figure see the Appendix A.

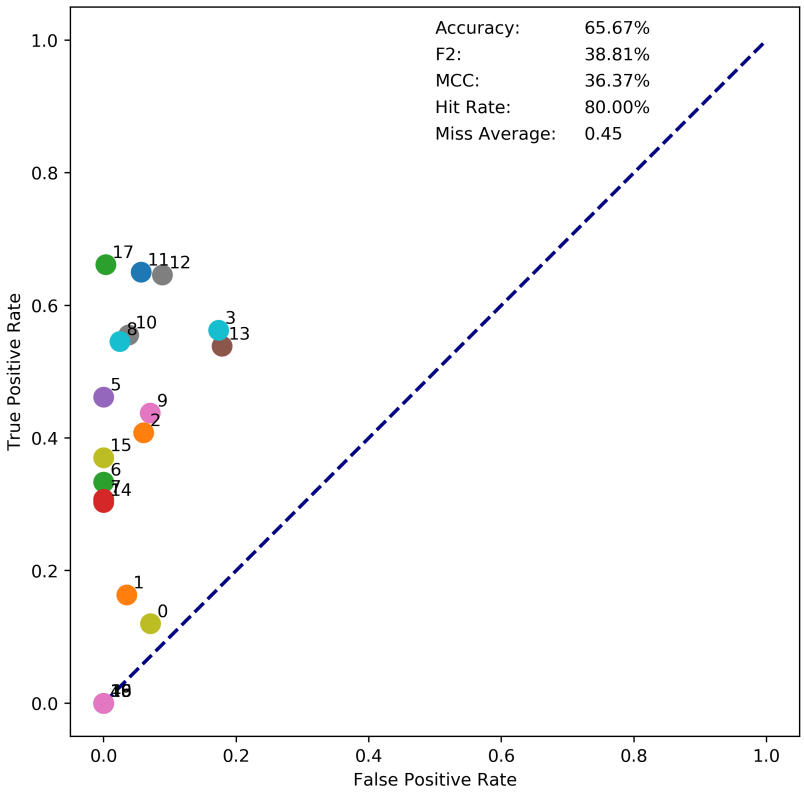



Figure 5.18: The CARLA swerve set has the same issues as the hardware set, but still performs well overall. For the full performance figure see the Appendix A.

The last two sets, being a mix, vary performance from video to video. However, it shows that the system can perform moderately well despite the differences found in the videos. It is important to note the largest difference between these sets and the previous is that more than one maneuver can occur per video and the ability to work continually is incredibly important.

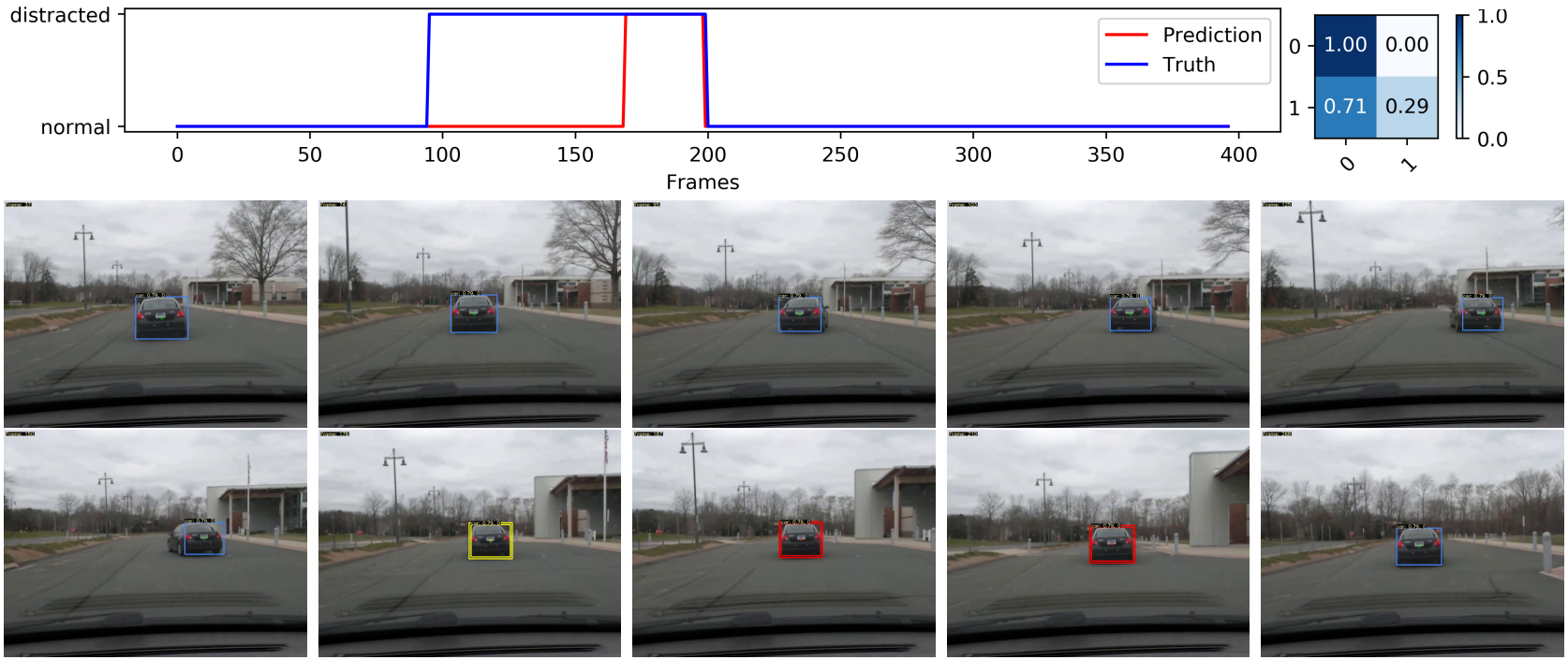



Figure 5.19: This video is an example of how the swerve maneuver is detected late, here one can see how the maneuver is only flagged near its completion.

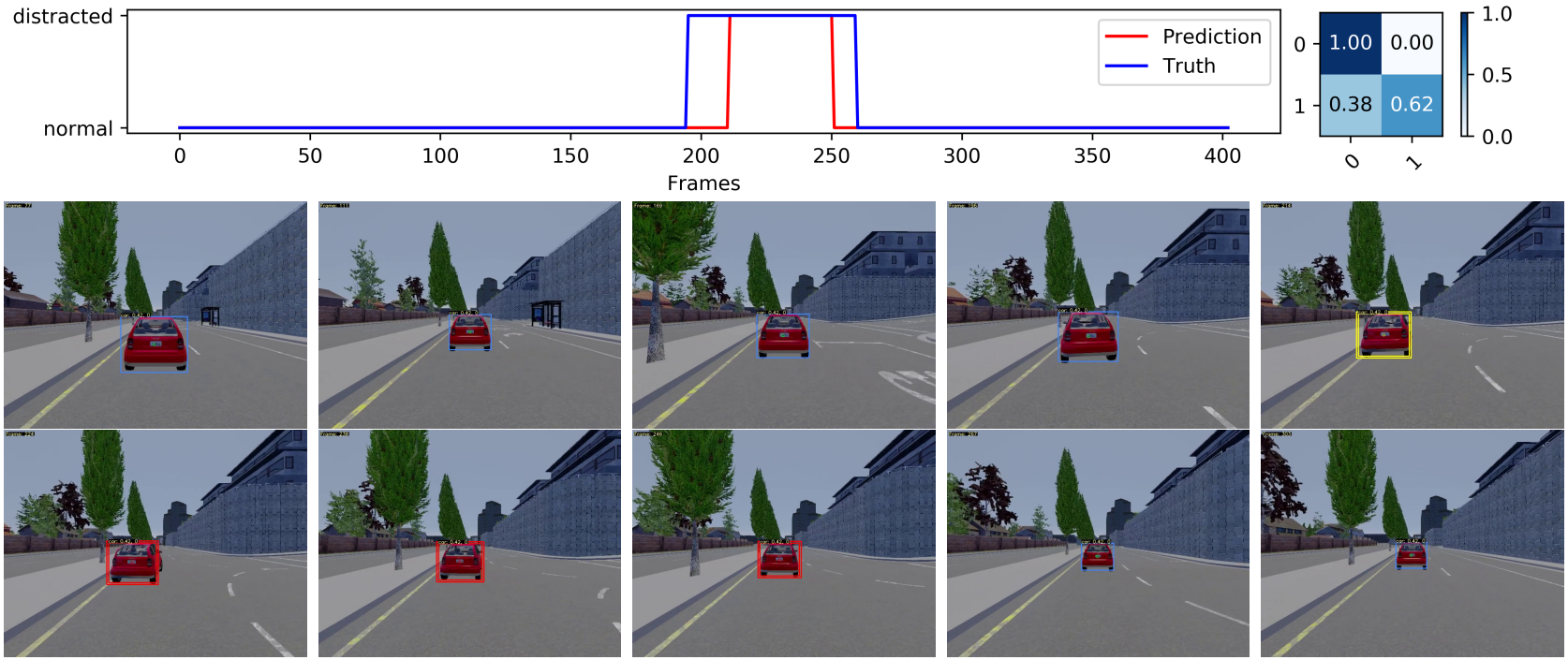



Figure 5.20: The detection is however, sometimes on time as seen in this simulator example. This is a video of an Audi A2 bumping up against the curb on the left and correcting.

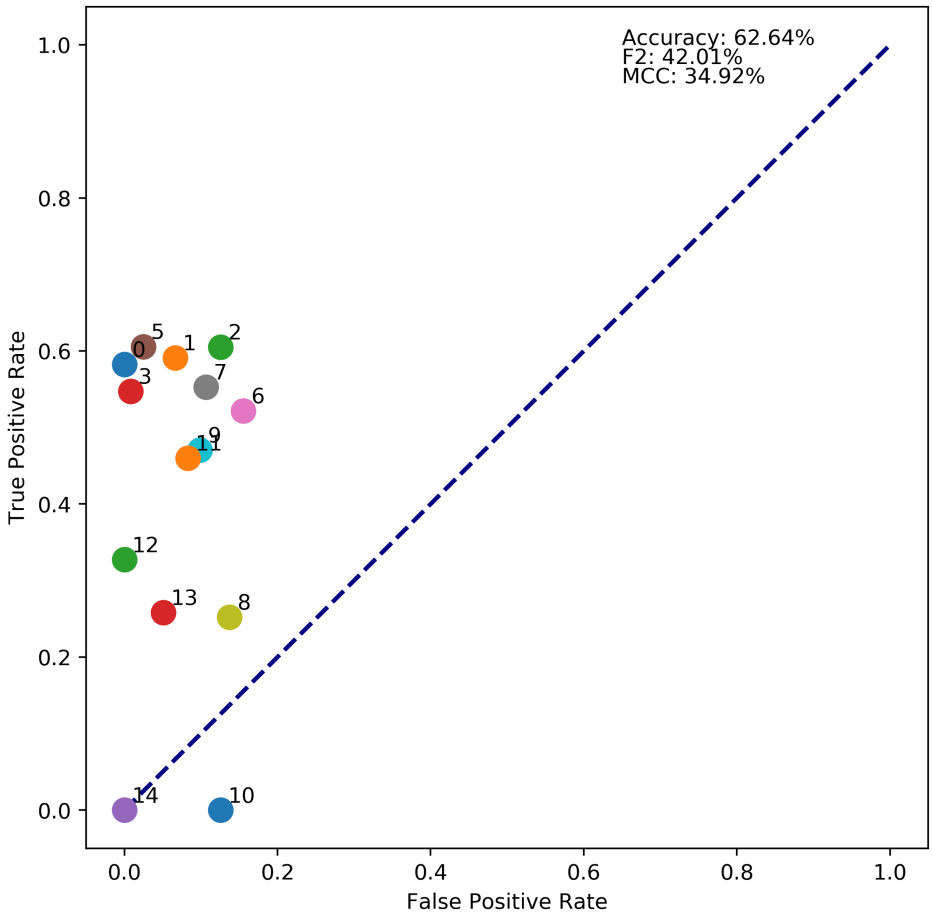



Figure 5.21: The ROC space for the hardware mix dataset contains a variety of videos and the dots here are not expected to group up. The overall performance on this set is promising and shows the system's ability to tag many of the maneuvers occurring in each video. For the full performance figure see the Appendix A.

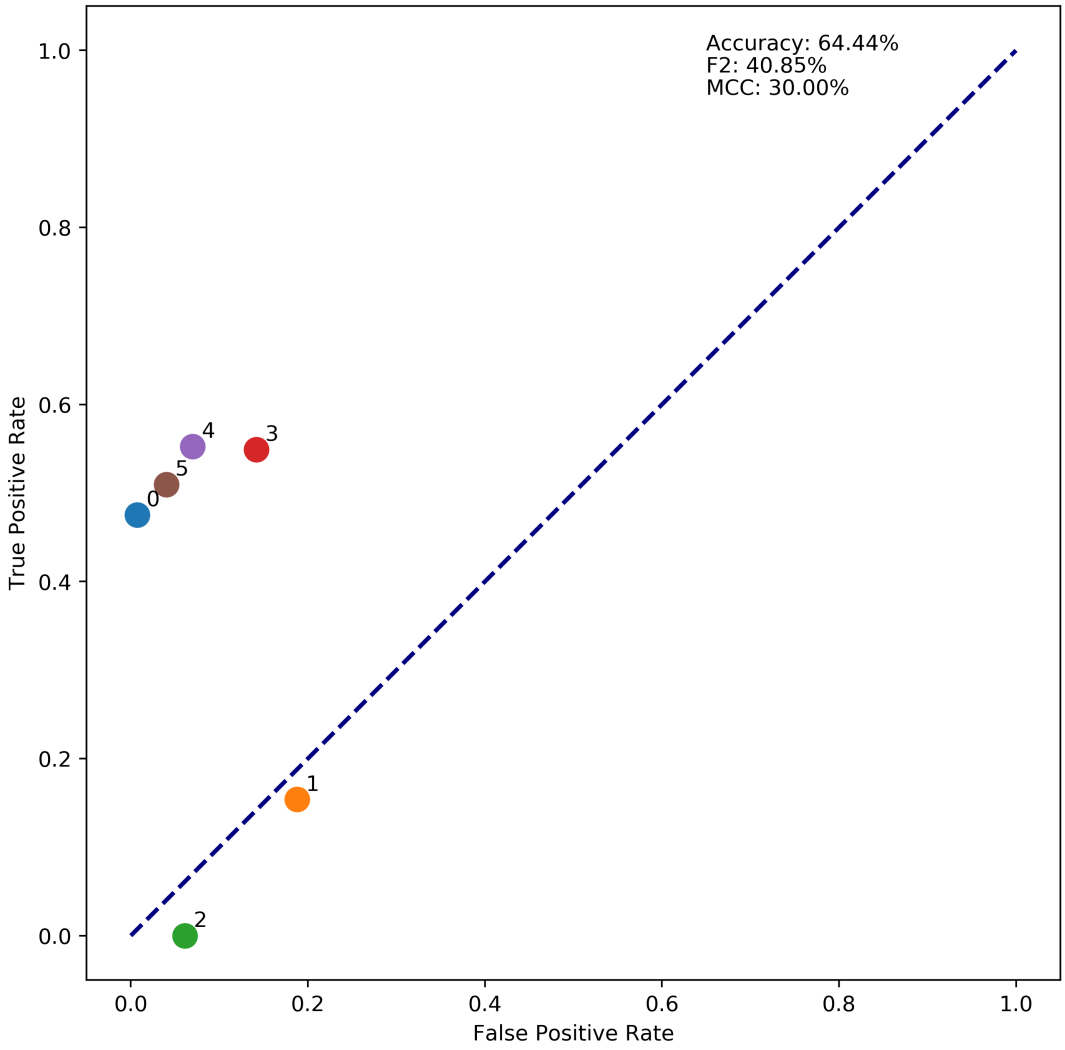



Figure 5.22: The YouTube dataset is small, and it was difficult to find videos which worked well with the detector setup as it was designed for small resolution video streams. It was however able to function in some cases even as the camera angle and setup varied between video. Although, no impressive results can be shown with this set. For the full performance figure see the Appendix A.

A notable highlight from this set is the system's ability to recognize distracted behavior without the anomaly score rising above the threshold. This occurs in the videos where a driver is distracted multiple times, the first is recognized by the anomaly detector and when

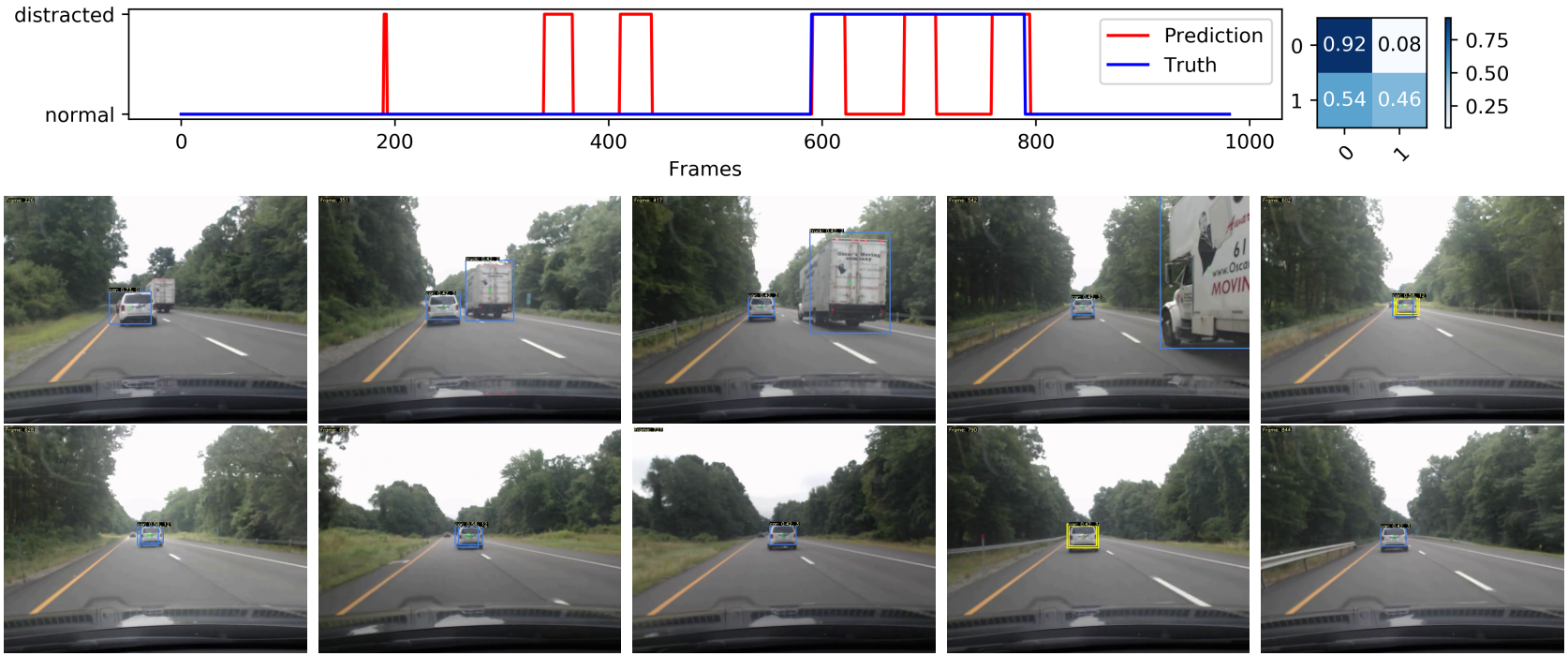



Figure 5.23: The video here shows a unique environment and a distracted driver recorded by the hardware system in the wild. The driver briefly drifts over the line, remains there for some time and drifts back. A few false positives are detected in the beginning as the vehicle is switching lanes and then true positives at the beginning and the end of its drift.

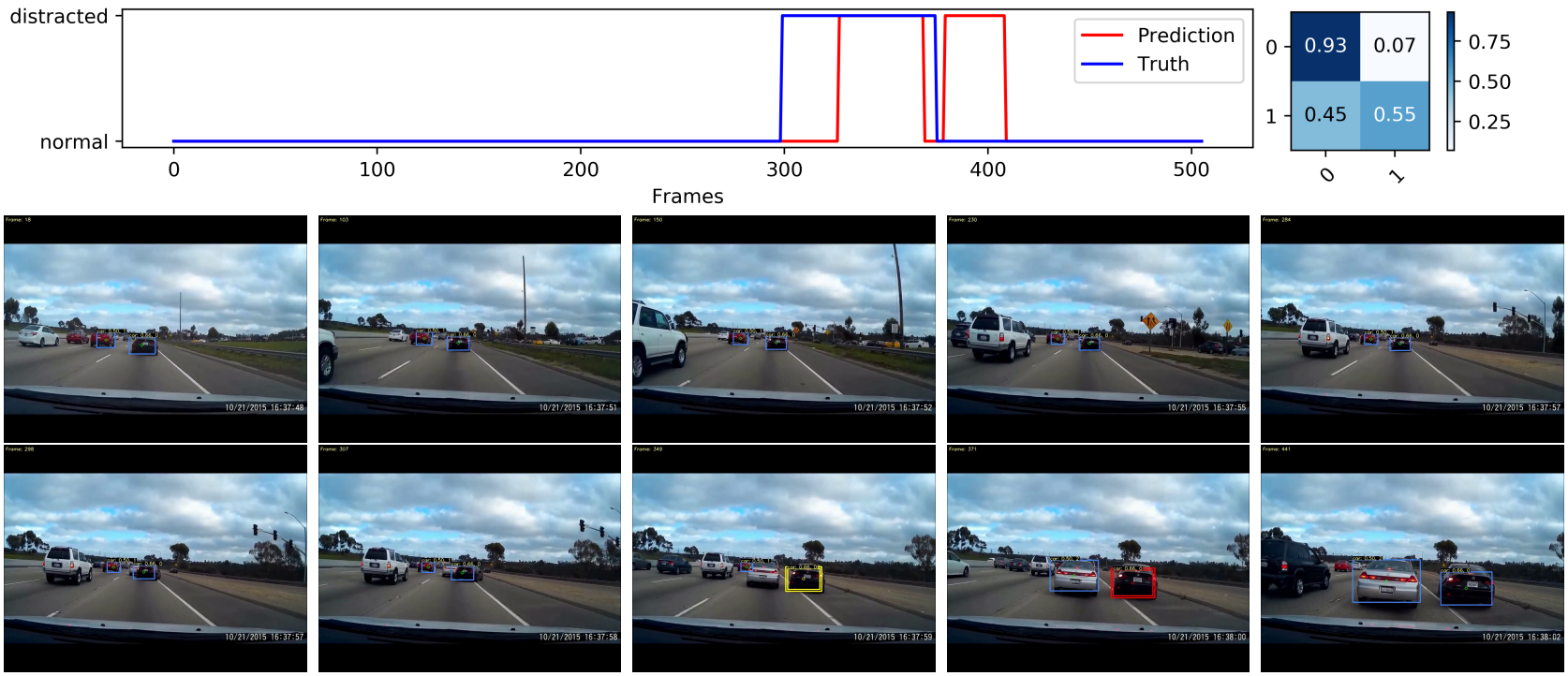



Figure 5.24: This YouTube video depicts a vehicle who is not paying attention as the car in front of them comes to a stop. It then swerves out of the way at the last second to avoid being hit. The system can detect this sudden movement and flag the driver.

it happens a second time it remembers what distracted behavior looks like and can flag the driver only using the cluster information as described in the previous section.

## 5.4 Summary

This chapter explored the effects of certain parameters and introduced the metrics used for performance. There is a direct trade off with the true positive rate and the false positive rate for some of the most important parameters. This trade of for any specific application can be optimized with the NeymanPearson lemma [79]. Over the 102 videos in the dataset the system was able to achieve an accuracy score of 65.5% an F2 score of 43.08% and an MCC of 36.07%. The application specific metrics as tested on the swerve and drift data sets have a total of a 87.5% hit rate and a 0.4375 miss average.

Chapter 6 will draw conclusions about the current system performance and future of this work.

Table 5.1: The table below provides an overview of the dataset performance.

| **Name** | **Video Count** | **Purpose** | **Metrics** |
|---|---|---|---|
| Hardware Drift | 20 | Repetitive testing of the same drift maneuver with variations in duration, severity and vehicle. | Hit Rate: 100.00%<br>Miss Average: 0.55<br>F2: 54.02%<br>MCC: 50.37%<br>Accuracy: 72.53% |
| CARLA Drift | 20 | Same as above, except with simulated videos as scripted with CARLA, provides randomness in all above areas with the addition of environment. | Hit Rate: 100.00%<br>Miss Average: 0.40<br>F2: 59.83%<br>MCC: 55.32%<br>Accuracy: 75.28% |
| Hardware Swerve | 20 | Repetitive testing of the same swerve maneuver with variations in duration, severity and vehicle. | Hit Rate: 70.00%<br>Miss Average: 0.35<br>F2: 29.13%<br>MCC: 12.39%<br>Accuracy: 53.62% |
| CARLA Swerve | 20 | Same as above, except with simulated videos as scripted with CARLA, provides randomness in all above areas with the addition of environment. | Hit Rate: 80.00%<br>Miss Average: 0.45<br>F2: 36.98%<br>MCC: 33.44%<br>Accuracy: 64.59% |
| Hardware Mixed | 15 | Provides a variety of maneuvers and environments, includes forward and rear facing videos. | F2: 42.01%<br>MCC: 34.92%<br>Accuracy: 62.64% |
| YouTube Mixed | 7 | Shows extreme driving episodes with different cameras and perspectives. | F2: 40.85%<br>MCC: 30.00%<br>Accuracy: 64.44% |

# Chapter 6

# Conclusion

The first contribution for this work is to provide a hardware test bed with adequate on-board processing to take in video, detect and track vehicles and preform general processing. The testbed that was chosen comprised of a Nvidia Jetson Nano and a Coral Edge TPU. The processing power proved adequate to track and estimate the behavior of one vehicle at an average of 30 FPS with a resolution of 800 by 600. As mentioned previously in Chapter 3 the performance does dip down to 15 FPS under difficult tracking situations, but the testbed succeeds in allowing the system to function in real world scenarios.

The second is to create an image and signal processing expert solution for detecting distracted driving artifacts. The expert solution provided consists primarily of an CNN based object detector a correlation-based tracker and an anomaly detection scoring system. This expert system was able to provide insight into the behavior and driving style of the vehicle in question when ran on videos. The output of the expert system is an anomaly score per track and when viewed alongside the video aids in locating unusual behavior.

Finally, an online machine learning framework was added to learn from and complement the expert system. The tracked vehicles features where decomposed into windowed sub-features and clustered to group driving behavior. This helped to associate the groups to erratic behavior by relating the score provided by the expert system and the label of the current cluster. Over the 102 videos in the dataset the system was able to achieve an accuracy score of 65.5% an F2 score of 43.08% and an MCC of 36.07%. The application

specific metrics as tested on the swerve and drift data sets have a total of a 87.5% hit rate and a 0.4375 miss average. As noted in the previous section these numbers aren't the exiting part of the contribution. Rather, providing a way to combine signal processing and machine learning to help give unsupervised learning algorithms innate knowledge about the problem they are trying to solve is.

## 6.1 Future Work

The system presented has laid the groundwork for many applications and further research. The three applications which brought on the inspiration of this work follow. First, to build a local vehicle communication network where the onboard behavior evaluation systems would work together to identify distracted or dangerous driving and warn drivers in the area. Next, to build a construction site monitoring system which looks out at oncoming cars and watches for distracted drivers to then alert the construction workers. Lastly, to aid in self-driving car planning decision making. The addition of behavior assessment could help to make marginally safer planning decisions.

In terms of extensions to this work, robustness could be added in the form of additional sensors and detectors to aid in preventing false positives. Lane marking could make the system work during turns. Internal measurement unit (IMU) and steering angle could aid to detect the path of the vehicle the system is mounted on and thus decorrelate its movement with the tracked vehicle. Blinker and brake light detection would add other metrics to asses behavior against. For example, a driver pauses an unusually long time at a stop sign or has a low reaction time when a traffic changes. Further parameter tuning of the current system could also bring forward better results.

# Bibliography

[1] N. C. for Statistics and Analysis, "Distracted driving in fatal crashes, 2017," *Traffic Safety Facts Research Note*, vol. DOT HS 812 700, April 2019. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812700

[2] ——, "Driver electronic device use in 2017," *Traffic Safety Facts Research Note*, vol. DOT HS 812 665, Jan 2019. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812665

[3] ——, "Distracted driving 2015," *Traffic Safety Facts Research Note*, vol. DOT HS 812 381, March 2017. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812381

[4] ——, "Distracted driving 2011," *Traffic Safety Facts Research Note*, vol. DOT HS 811 737, April 2013. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/811737

[5] ——, "An examination of driver distraction as recorded in nhtsa databases," *Traffic Safety Facts Research Note*, vol. DOT HS 811 216, Sept 2009. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/811216

[6] (2019) Car Drivers - Statistics and Facts. [Online]. Available: https://www.statista.com/topics/1197/car-drivers/

[7] (2019) 2019 Distracted Driving Statistics. [Online]. Available: https://www.thezebra.com/distracted-driving-statistics/

[8] D. Stavrinos, J. L. Jones, A. A. Garner, R. Griffin, C. A. Franklin, D. Ball, S. C. Welburn, K. K. Ball, V. P. Sisiopiku, and P. R. Fine, "Impact of distracted driving on safety and traffic flow," *Accident; analysis and prevention*, vol. 61, no. 3, pp. 63–70, Feb 2013. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4435680/

[9] L. Fridman, P. Langhans, J. Lee, and B. Reimer, "Driver gaze estimation without using eye movement," *CoRR*, vol. abs/1507.04760, 2015. [Online]. Available: http://arxiv.org/abs/1507.04760

[10] C. Marina Martinez, M. Heucke, F. Wang, B. Gao, and D. Cao, "Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 3, pp. 666–676, March 2018.

[11] L. Fridman, "Human-centered autonomous vehicle systems: Principles of effective shared autonomy," *CoRR*, vol. abs/1810.01835, 2018. [Online]. Available: http://arxiv.org/abs/1810.01835

[12] A. Eskandarian and A. Mortazavi, "Evaluation of a smart algorithm for commercial vehicle driver drowsiness detection," in *2007 IEEE Intelligent Vehicles Symposium*, June 2007, pp. 553–559.

[13] J. Pohl, W. Birk, and L. Westervall, "A driver-distraction-based lane-keeping assistance system," *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, vol. 221, no. 4, pp. 541–552, 2007. [Online]. Available: https://doi.org/10.1243/09596518JSCE218

[14] V. Vaitkus, P. Lengvenis, and G. ylius, "Driving style classification using long-term accelerometer information," in *2014 19th International Conference on Methods and Models in Automation and Robotics (MMAR)*, Sep. 2014, pp. 641–644.

[15] N. Kuge, T. Yamamura, O. Shimoyama, and A. Liu, "A driver behavior recognition method based on a driver model framework," *Society of Automotive Engineers,*

*Inc.*, vol. 2000-01-0349, 1998. [Online]. Available: https://web.mit.edu/people/amliu/Papers/SAE2000_Kuge.pdf

[16] (2019) The new standard for fleet safety is here. [Online]. Available: https://www.nauto.com/product

[17] V. Ramanishka, Y. Chen, T. Misu, and K. Saenko, "Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning," *CoRR*, vol. abs/1811.02307, 2018. [Online]. Available: http://arxiv.org/abs/1811.02307

[18] N. Deo and M. M. Trivedi, "Multi-modal trajectory prediction of surrounding vehicles with maneuver based lstms," *CoRR*, vol. abs/1805.05499, 2018. [Online]. Available: http://arxiv.org/abs/1805.05499

[19] Christopher P. Ricci, "Behavioral Tracking and Vehicle Applications," United States Patent 9,296,299, Mar. 29, 2016.

[20] Z. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *CoRR*, vol. abs/1807.05511, 2018. [Online]. Available: http://arxiv.org/abs/1807.05511

[21] L. Leal-Taixé, A. Milan, K. Schindler, D. Cremers, I. D. Reid, and S. Roth, "Tracking the trackers: An analysis of the state of the art in multiple object tracking," *CoRR*, vol. abs/1704.02781, 2017. [Online]. Available: http://arxiv.org/abs/1704.02781

[22] (2018) Self-driving cars are headed toward an AI road-block. [Online]. Available: https://www.theverge.com/2018/7/3/17530232/self-driving-ai-winter-full-autonomy-waymo-tesla-uber

[23] LeCun, Y. and Haffner, P. and Bottou, L. and Bengio, Y., "Object Recognition with Gradient-Based Learning," in *Feature Grouping*, Forsyth, D., Ed. Springer, 1999.

[24] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[26] (2018) MobileNet Version 2. [Online]. Available: https://machinethink.net/blog/mobilenet-v2/

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[28] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[29] (2019) Models Built for the Edge TPU. [Online]. Available: https://coral.withgoogle.com/models/

[30] (2019) Run a model on the Edge TPU. [Online]. Available: https://coral.withgoogle.com/docs/dev-board/get-started/#run-a-model-on-the-edge-tpu

[31] (2017) Object Tracking using OpenCV . [Online]. Available: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/

[32] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[33] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters." Conference on Computer Vision and Pattern Recognition, 6 2010.

[34] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.

[35] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg, "Accurate scale estimation for robust visual tracking," *ACCURATE SCALE ESTIMATION FOR ROBUST VISUAL TRACKING*, pp. 1–11, 2014.

[36] O. M. Essenwanger, *Elements of statistical analysis*. Elsevier, 1986.

[37] (2013) Hann Window Image. [Online]. Available: https://en.wikipedia.org/wiki/Window_function#/media/File:Window_function_and_its_Fourier_transform_%E2%80%93_Hann_(n_=_0...N).svg

[38] A. Antoniadis, J. Bigot, and S. Lambert-Lacroix, "Peaks detection and alignment for mass spectrometry data," *Journal de la Société Française de Statistique*, vol. 151, no. 1, pp. 17–37, 2010.

[39] G. Galvas, "Time series forecasting used for real-time anomaly detection on websites author :," 2016.

[40] S. Agrawal and J. Agrawal, "Survey on anomaly detection using data mining techniques," *Procedia Computer Science*, vol. 60, pp. 708–713, 2015.

[41] (2014) Robust Peak Detection Algorithm (using z-scores). [Online]. Available: https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-data/22640362#22640362

[42] A. Khandakar, M. E. Chowdhury, R. Ahmed, A. Dhib, M. Mohammed, N. A. Al-Emadi, and D. Michelson, "Portable system for monitoring and controlling driver behavior and the use of a mobile phone while driving," *Sensors*, vol. 19, no. 7, p. 1563, 2019.

[43] M. C. Catalbas, T. Cegovnik, J. Sodnik, and A. Gulten, "Driver fatigue detection based on saccadic eye movements," in *2017 10th International Conference on Electrical and Electronics Engineering (ELECO)*, Nov 2017, pp. 913–917.

[44] M. Heideman, D. Johnson, and C. Burrus, "Gauss and the history of the fast fourier transform," *IEEE ASSP Magazine*, vol. 1, no. 4, pp. 14–21, October 1984.

[45] C. H. Lubba, S. S. Sethi, P. Knaute, S. R. Schultz, B. D. Fulcher, and N. S. Jones, "catch22: Canonical time-series characteristics," *CoRR*, vol. abs/1901.10200, 2019. [Online]. Available: http://arxiv.org/abs/1901.10200

[46] B. D. Fulcher and N. S. Jones, "hctsa: A computational framework for automated time-series phenotyping using massive feature extraction," *Cell systems*, vol. 5, no. 5, pp. 527–531, 2017.

[47] J. Shlens, "A tutorial on principal component analysis," *CoRR*, vol. abs/1404.1100, 2014. [Online]. Available: http://arxiv.org/abs/1404.1100

[48] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[49] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *arXiv preprint arXiv:1109.2378*, 2011.

[50] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979. [Online]. Available: http://www.jstor.org/stable/2346830

[51] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, Jun. 1999. [Online]. Available: http://doi.acm.org/10.1145/304181.304187

[52] W. Wang, J. Yang, R. Muntz *et al.*, "Sting: A statistical information grid approach to spatial data mining," in *VLDB*, vol. 97, 1997, pp. 186–195.

[53] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 19:1–19:21, Jul. 2017. [Online]. Available: http://doi.acm.org/10.1145/3068335

[54] D. Sculley, "Web-scale k-means clustering," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 1177–1178.

[55] (2019) Edge TPU Devices. [Online]. Available: https://aiyprojects.withgoogle.com/edge-tpu/

[56] (2019) Jetson Nano Developer Kit. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[57] (2019) Jetson Nano Brings AI Computing to Everyone. [Online]. Available: https://devblogs.nvidia.com/jetson-nano-ai-computing/

[58] (2017) Jetson TX2 Module. [Online]. Available: https://developer.nvidia.com/embedded/jetson-tx2

[59] (2018) Raspberry Pi 3 Model B. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[60] (2018) Intel Neural Compute Stick 2. [Online]. Available: https://software.intel.com/en-us/neural-compute-stick

[61] (2019) TPU Model Requirements. [Online]. Available: https://coral.withgoogle.com/docs/edgetpu/models-intro/#model-requirements

[62] (2017) Cortex-A57. [Online]. Available: https://developer.arm.com/ip-products/processors/cortex-a/cortex-a57

[63] (2014) Maxwell: The Most Advanced CUDA GPU Ever Made. [Online]. Available: https://devblogs.nvidia.com/maxwell-most-advanced-cuda-gpu-ever-made/

[64] P. Finnegan and P. Green, "The time to change lanes: A literature review," Tech. Rep., 1990.

[65] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the 23th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ser. ASPLOS-23. Williamsburg, VA, USA: ACM, 2018.

[66] (2012) Logitec C920. [Online]. Available: https://www.logitech.com/assets/45920/6/hd-pro-webcam-c920-quick-start-guide.pdf

[67] (2019) Car: By Pablo Rozenberg, CA. [Online]. Available: https://thenounproject.com/pabslabs/

[68] (2019) Threading Thread-based Parallelism. [Online]. Available: https://docs.python.org/3/library/threading.html

[69] (2019) Multiprocessing Process-based Parallelism. [Online]. Available: https://docs.python.org/3/library/multiprocessing.html

[70] (2018) Simple Object Tracking with OpenCV. [Online]. Available: https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

[71] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[72] (2019) YouTube. [Online]. Available: https://www.youtube.com/

[73] C. E. Metz, "Basic principles of roc analysis," in *Seminars in nuclear medicine*, vol. 8, no. 4. Elsevier, 1978, pp. 283–298.

[74] B. W. Matthews, "Comparison of the predicted and observed secondary structure of t4 phage lysozyme," *Biochimica et Biophysica Acta (BBA)-Protein Structure*, vol. 405, no. 2, pp. 442–451, 1975.

[75] Y. Sasaki *et al.*, "The truth of the f-measure," *Teach Tutor mater*, vol. 1, no. 5, pp. 1–5, 2007.

[76] (2009) ROC Plot. [Online]. Available: https://en.wikipedia.org/wiki/Receiver_operating_characteristic#/media/File:ROC_space-2.png

[77] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[78] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[79] J. Neyman and E. S. Pearson, "On the problem of the most efficient tests of statistical hypotheses," *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 231, pp. 289–337, 1933. [Online]. Available: http://www.jstor.org/stable/91247

# Appendix A

# Performance Reports

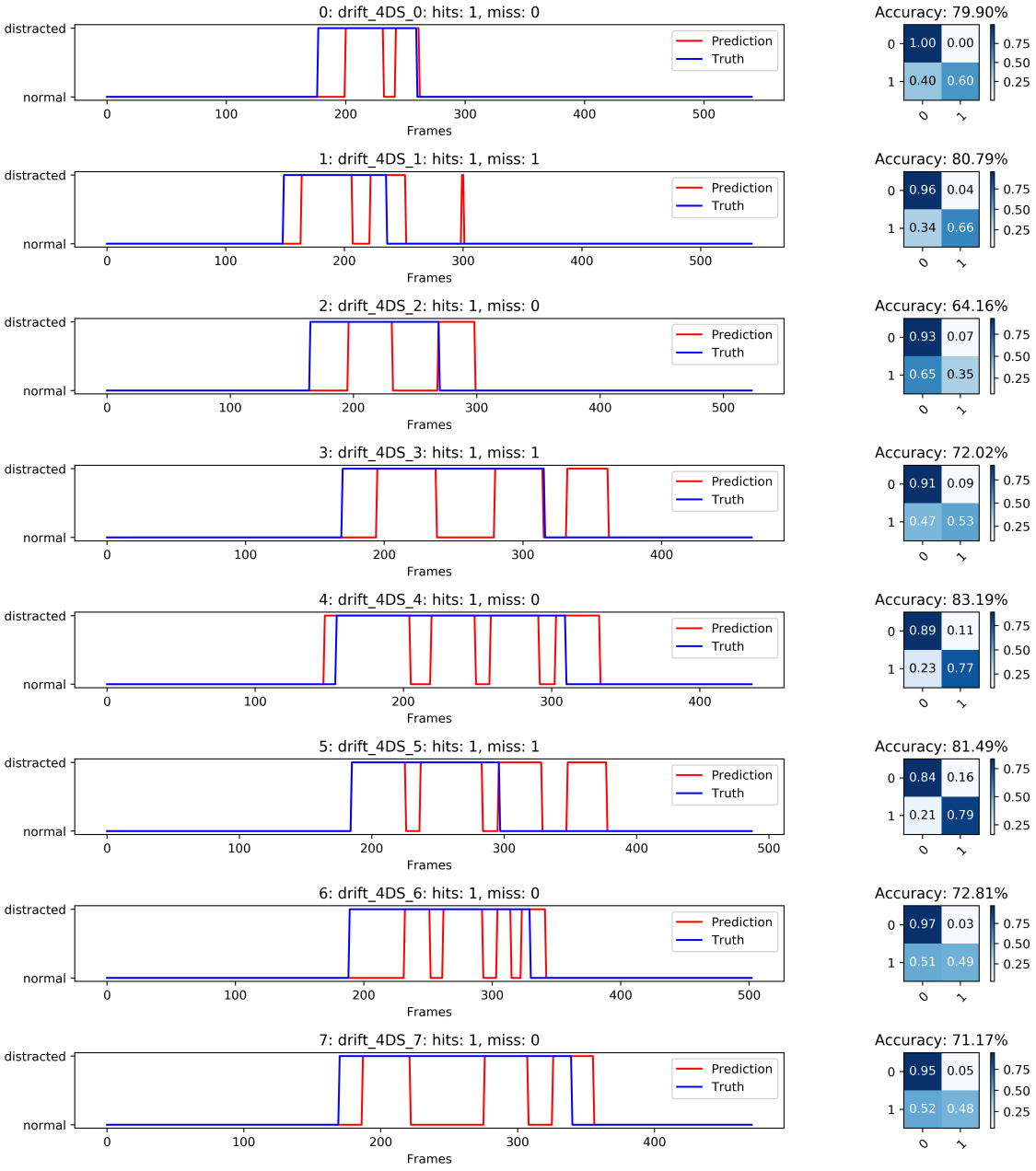



Figure A.1: Hardware Drift Batch Part 1 of 3

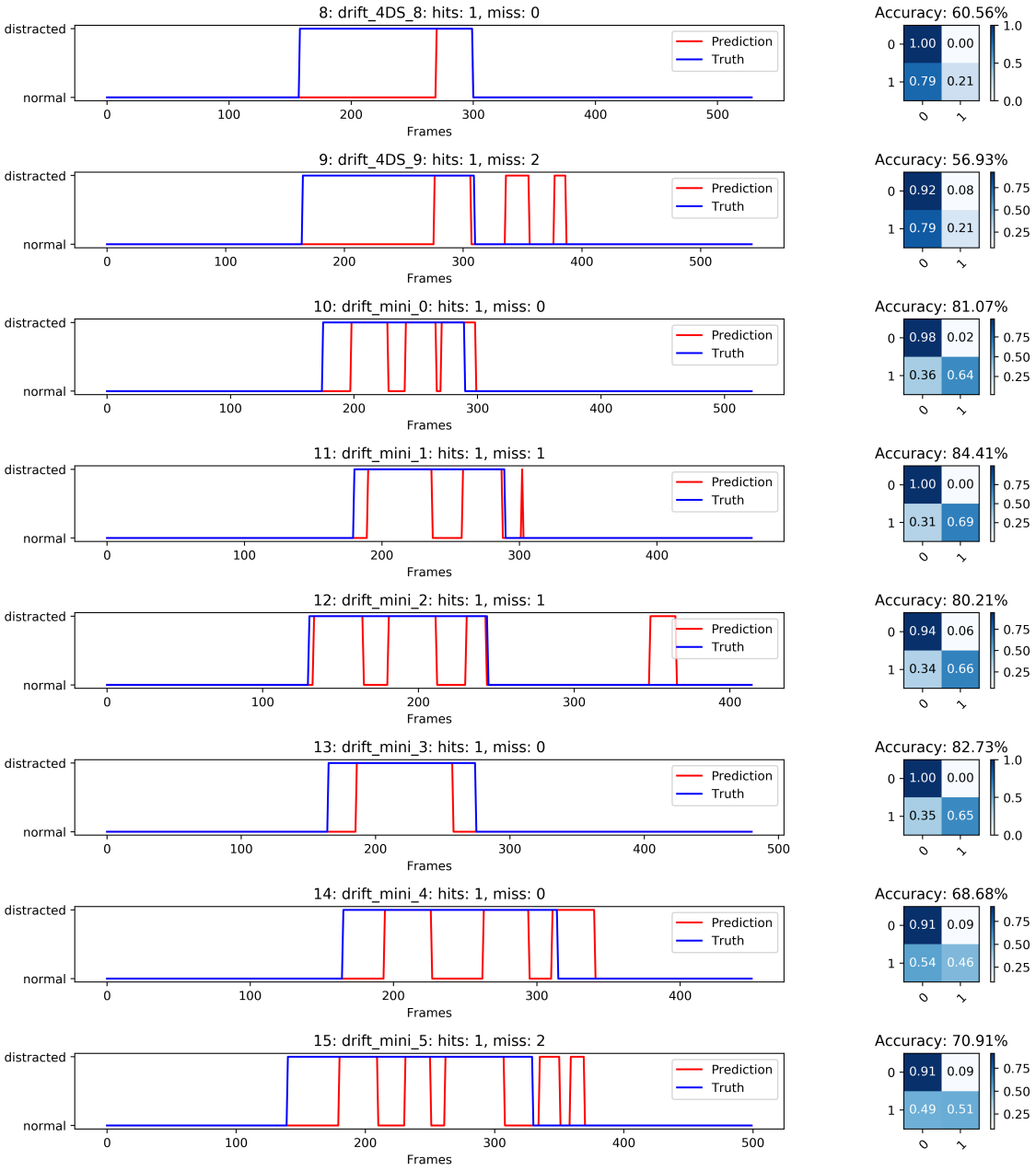



Figure A.2: Hardware Drift Batch Part 2 of 3

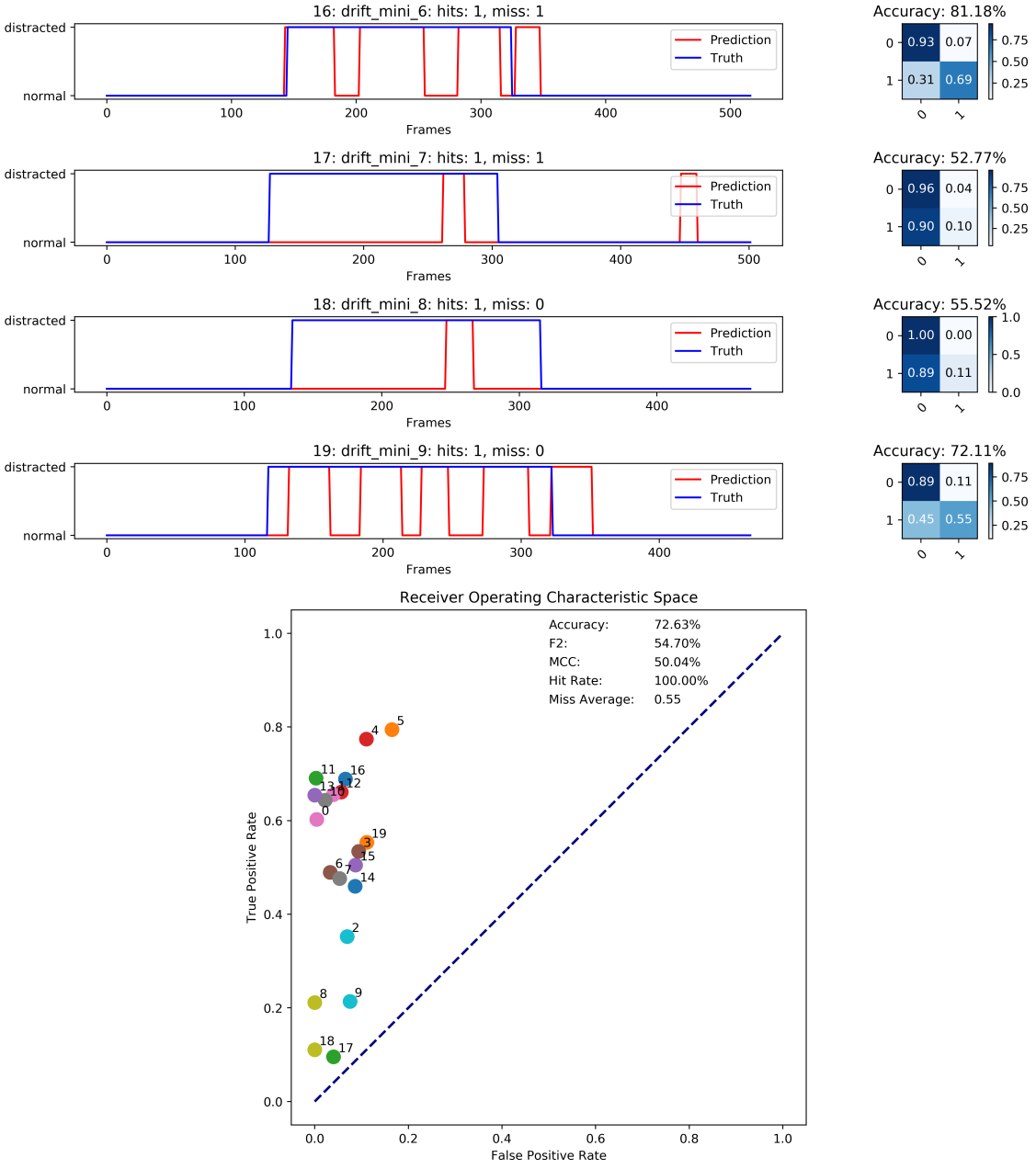



Figure A.3: Hardware Drift Batch Part 3 of 3

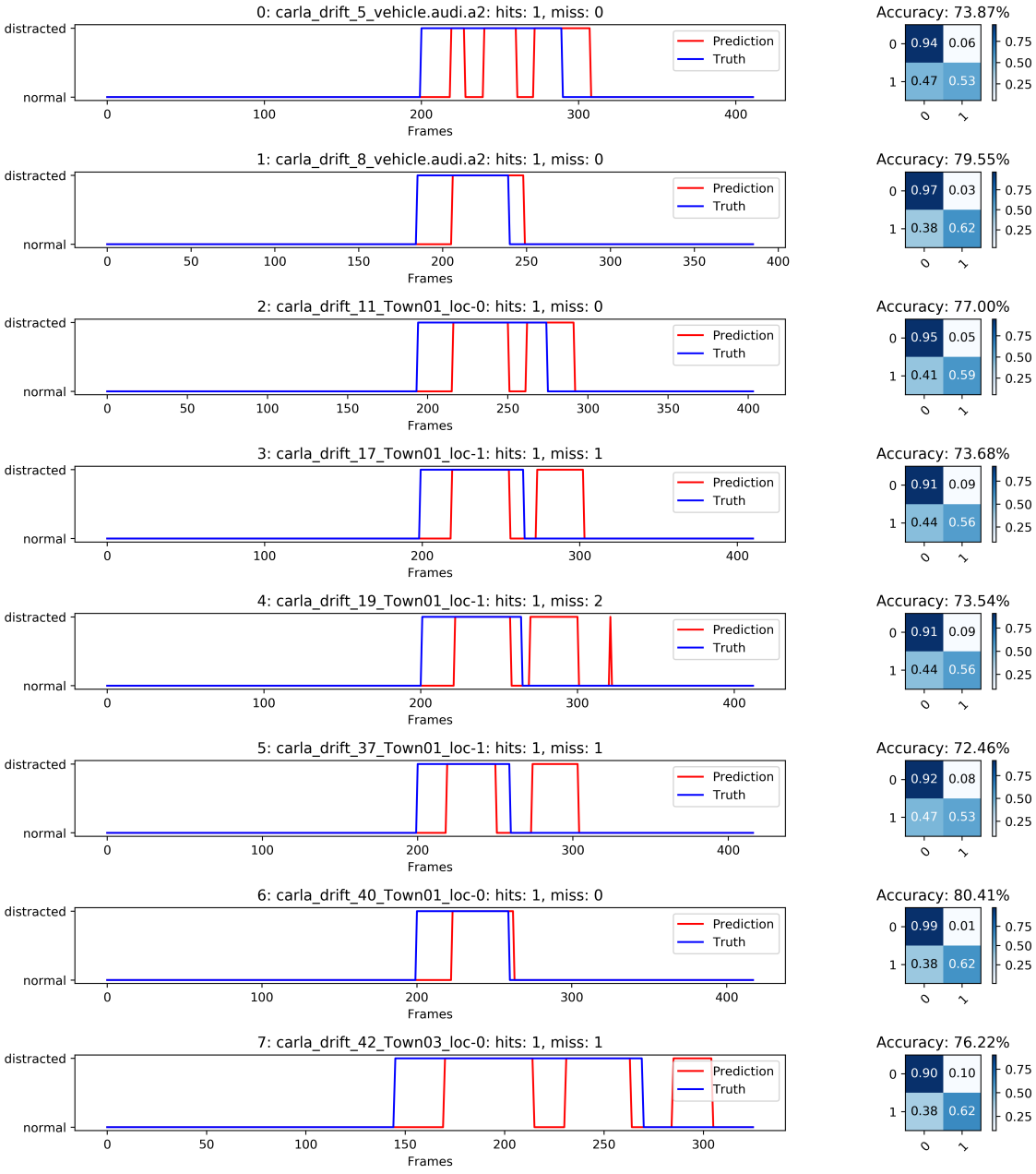



Figure A.4: CARLA Drift Batch Part 1 of 3

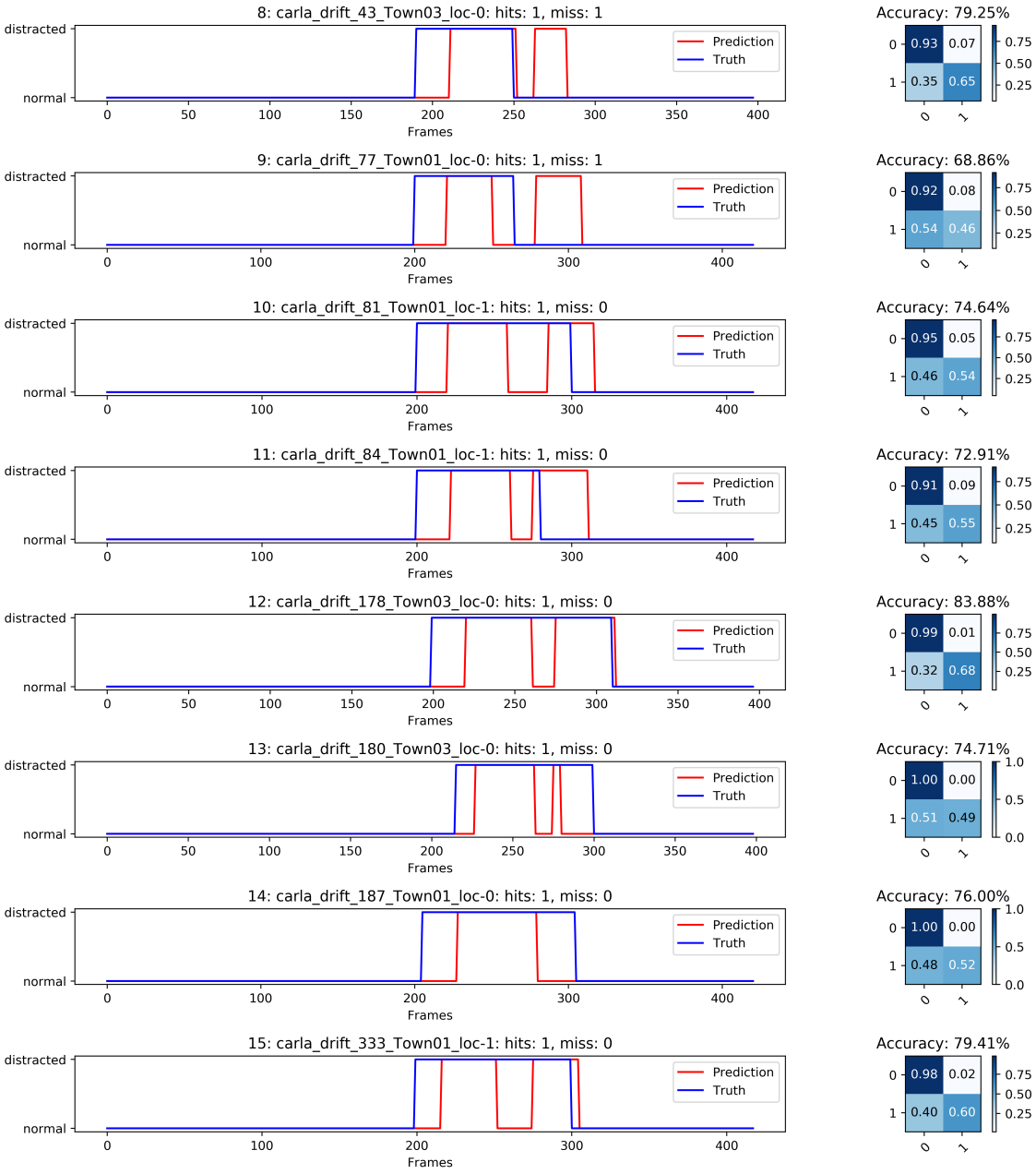



Figure A.5: CARLA Drift Batch Part 2 of 3

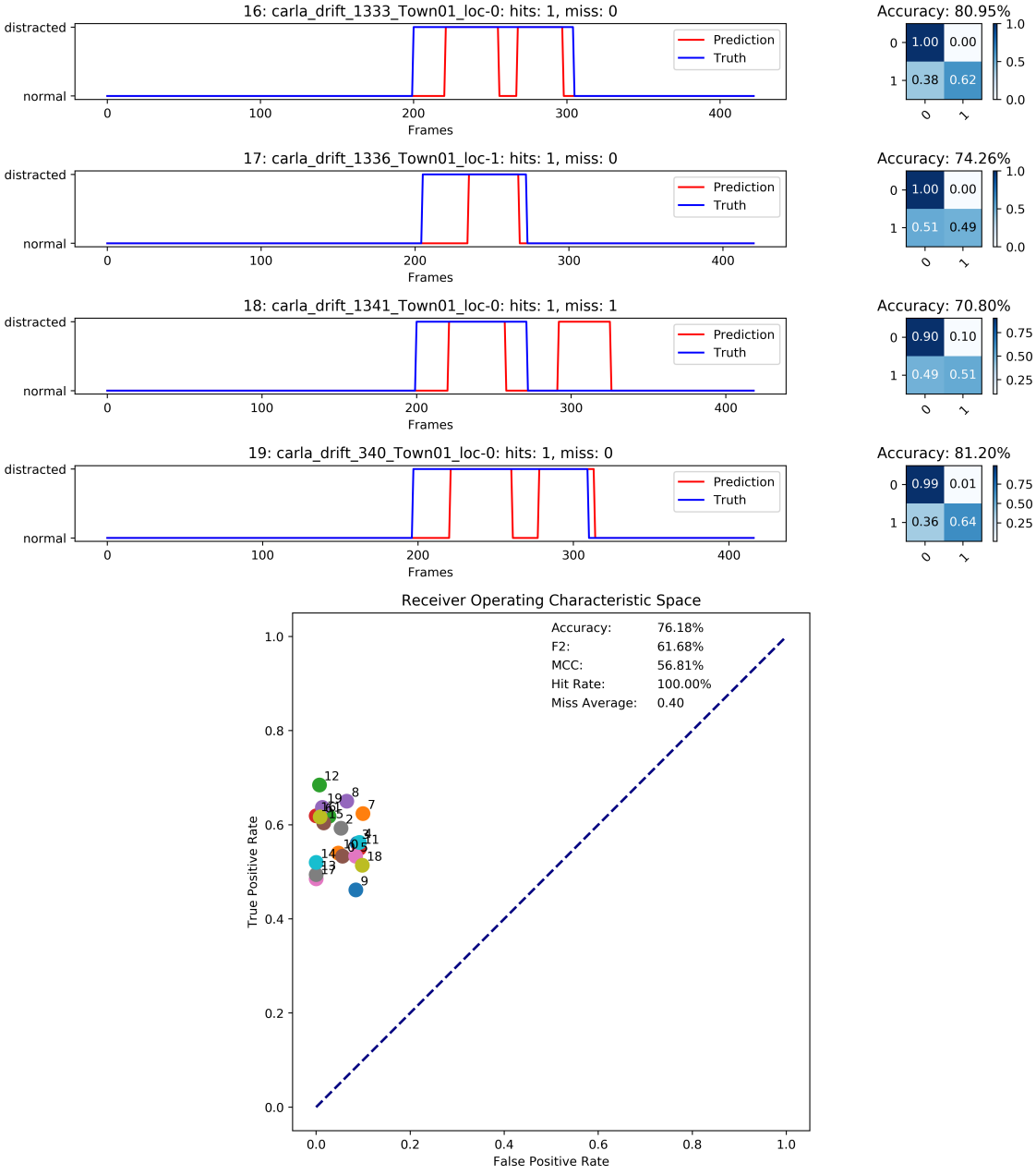



Figure A.6: CARLA Drift Batch Part 3 of 3

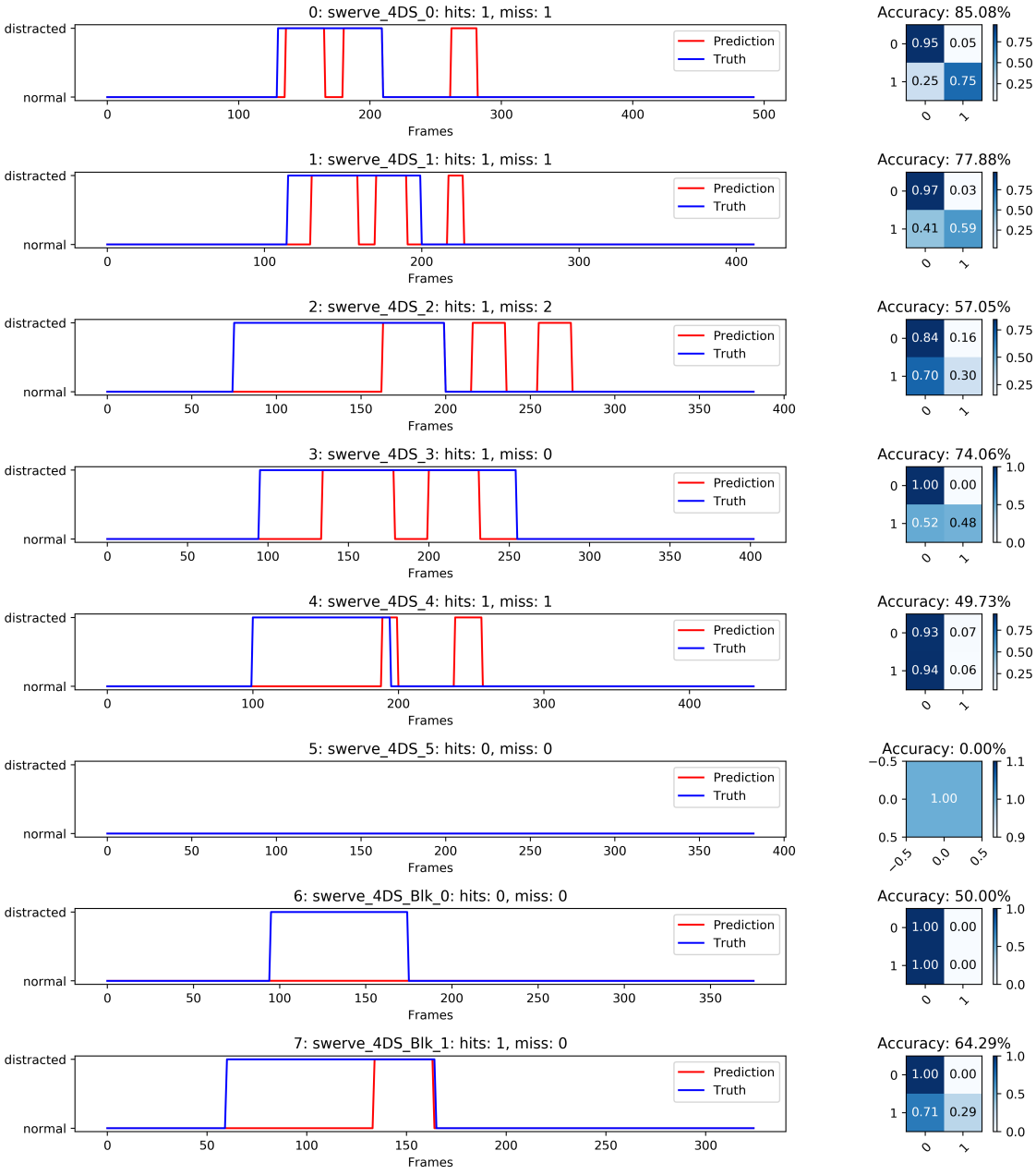



Figure A.7: Hardware Swerve Batch Part 1 of 3

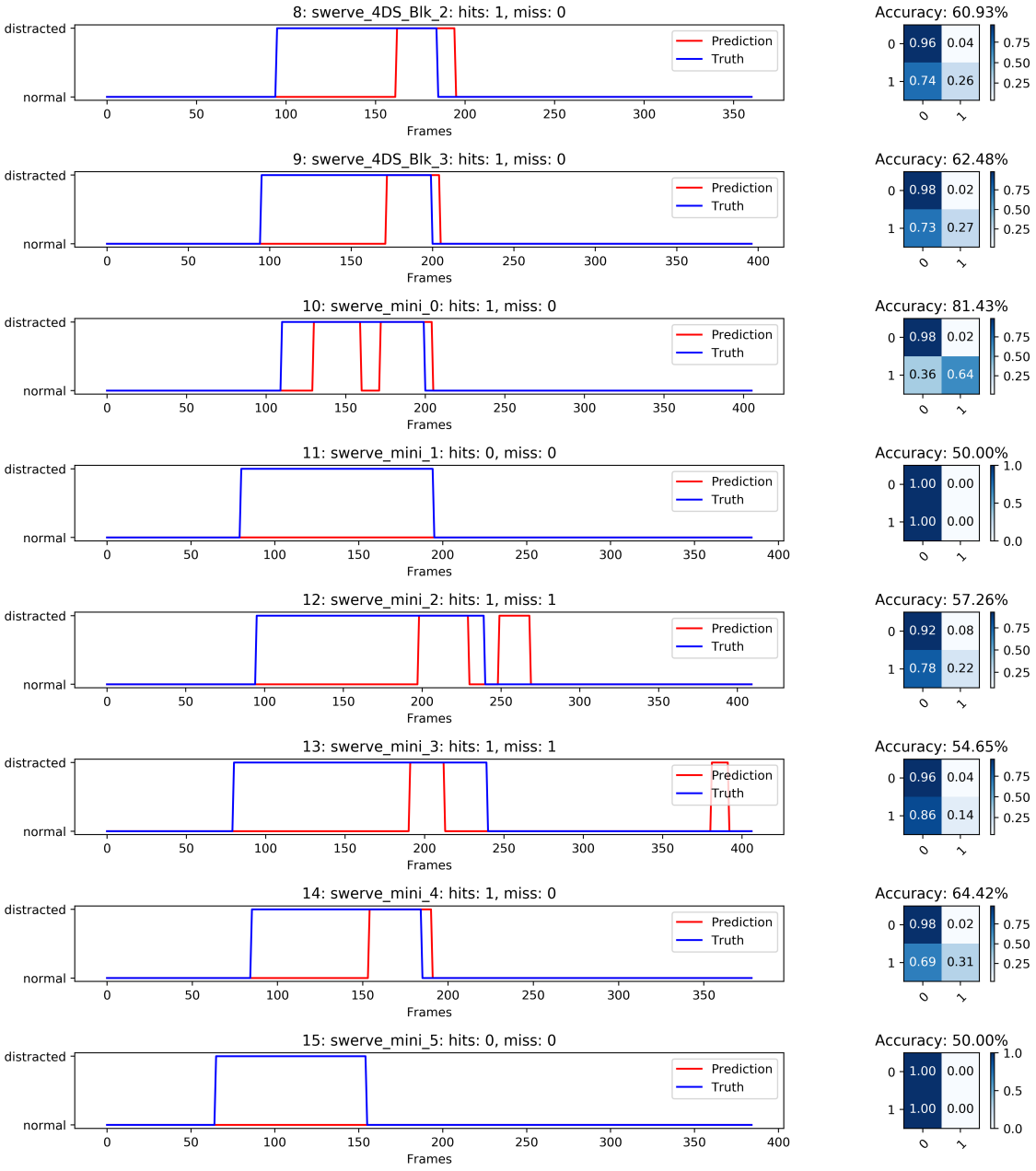



Figure A.8: Hardware Swerve Batch Part 2 of 3

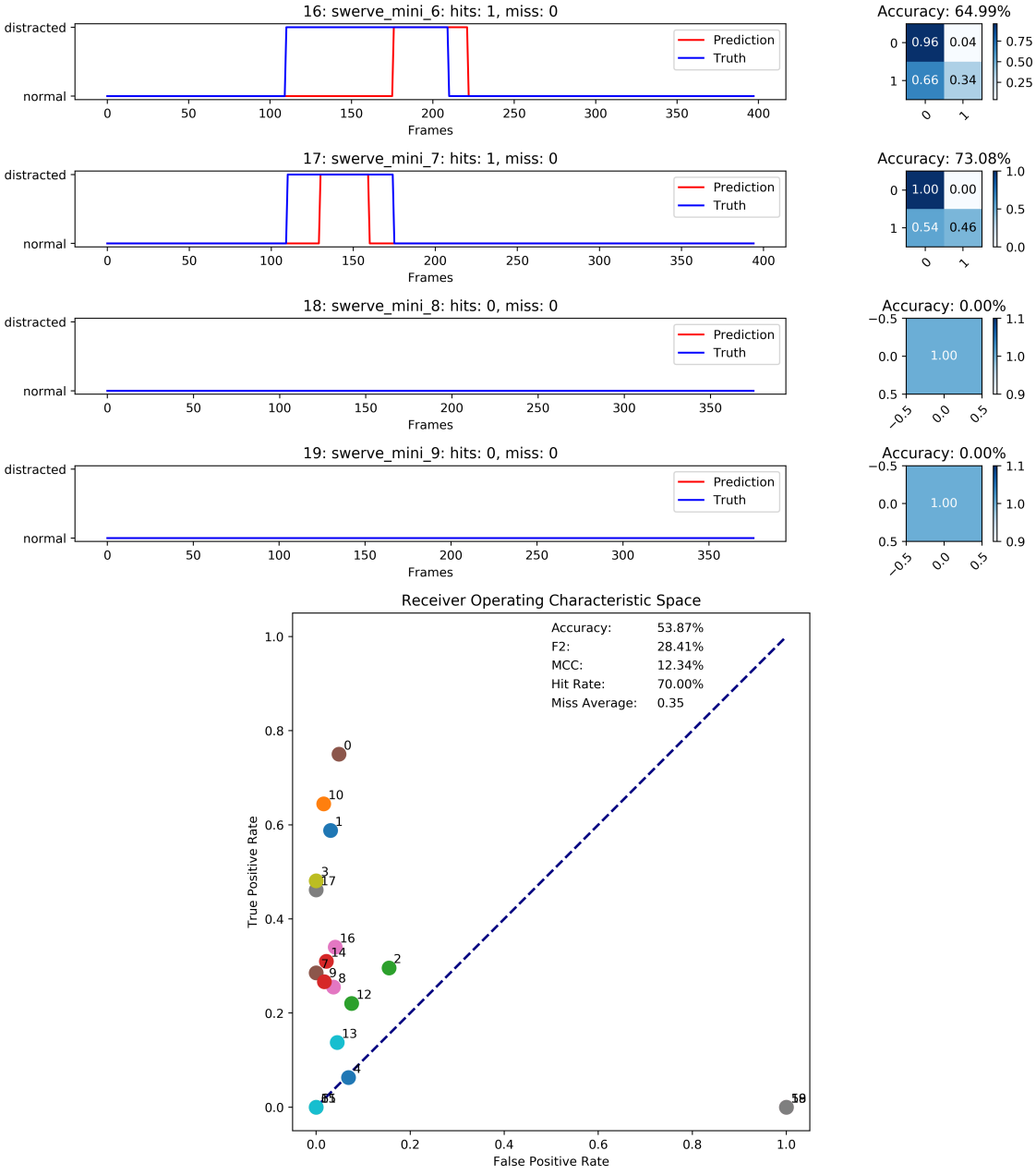



Figure A.9: Hardware Swerve Batch Part 3 of 3

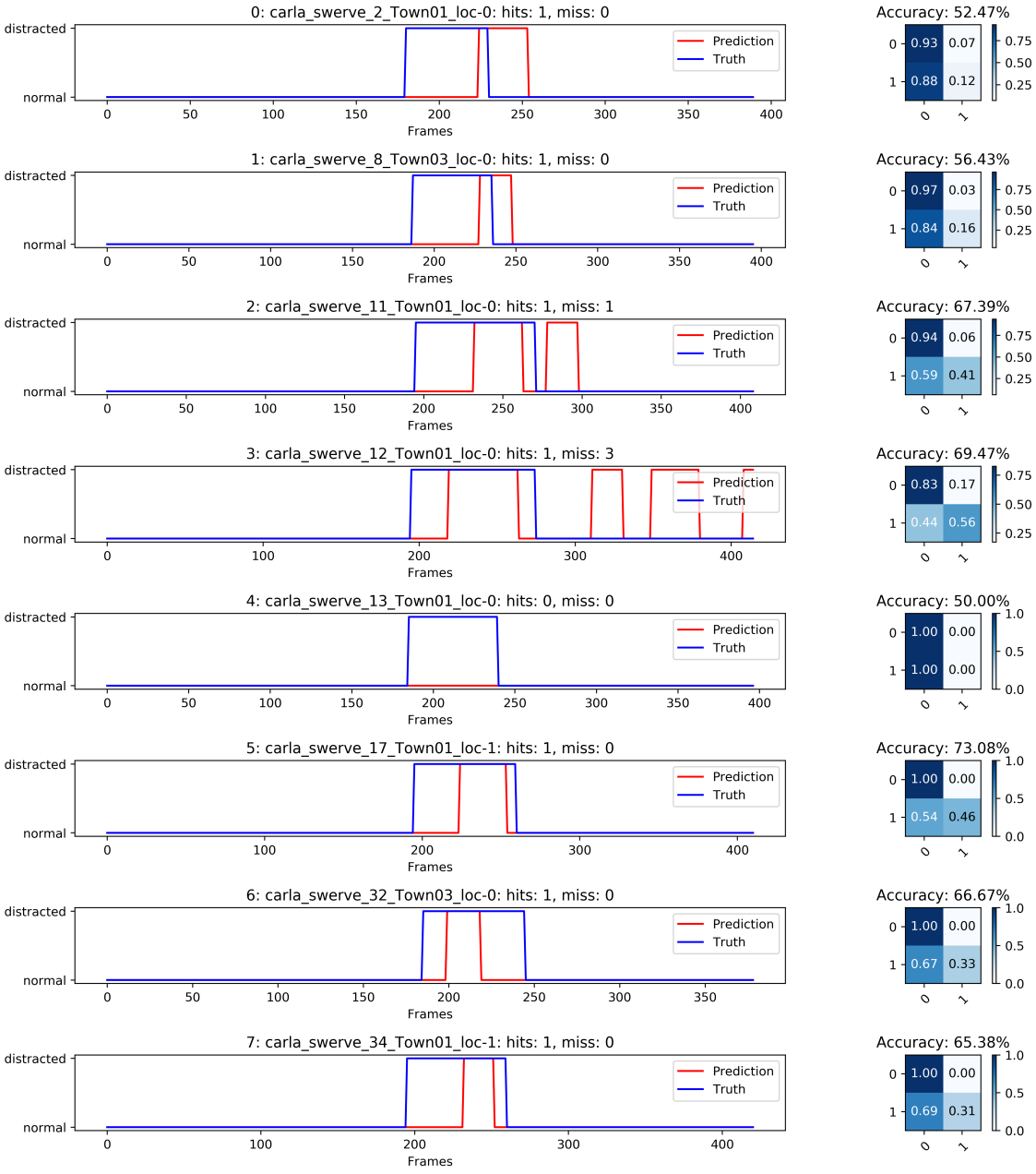



Figure A.10: CARLA Swerve Batch Part 1 of 3

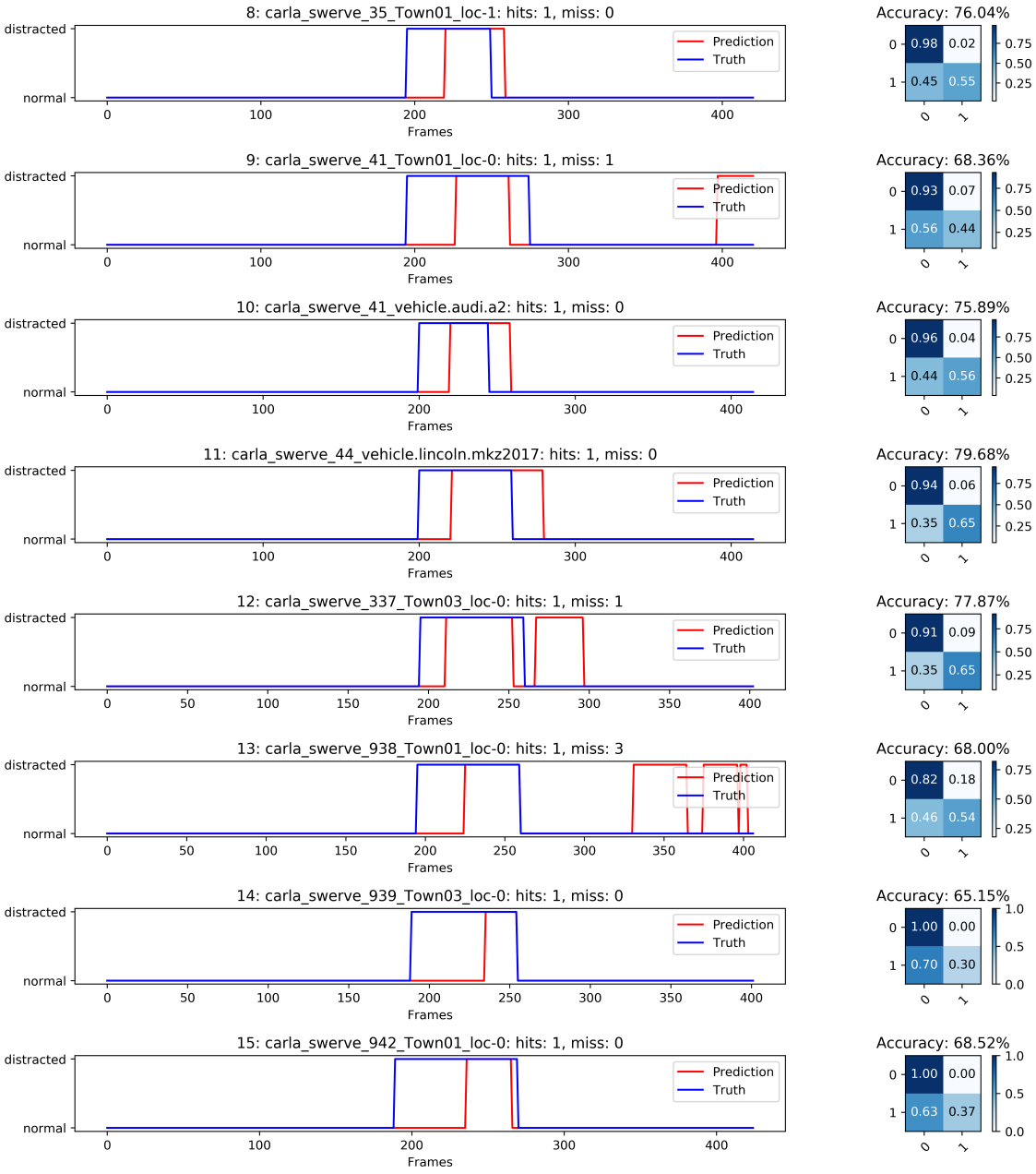



Figure A.11: CARLA Swerve Batch Part 2 of 3

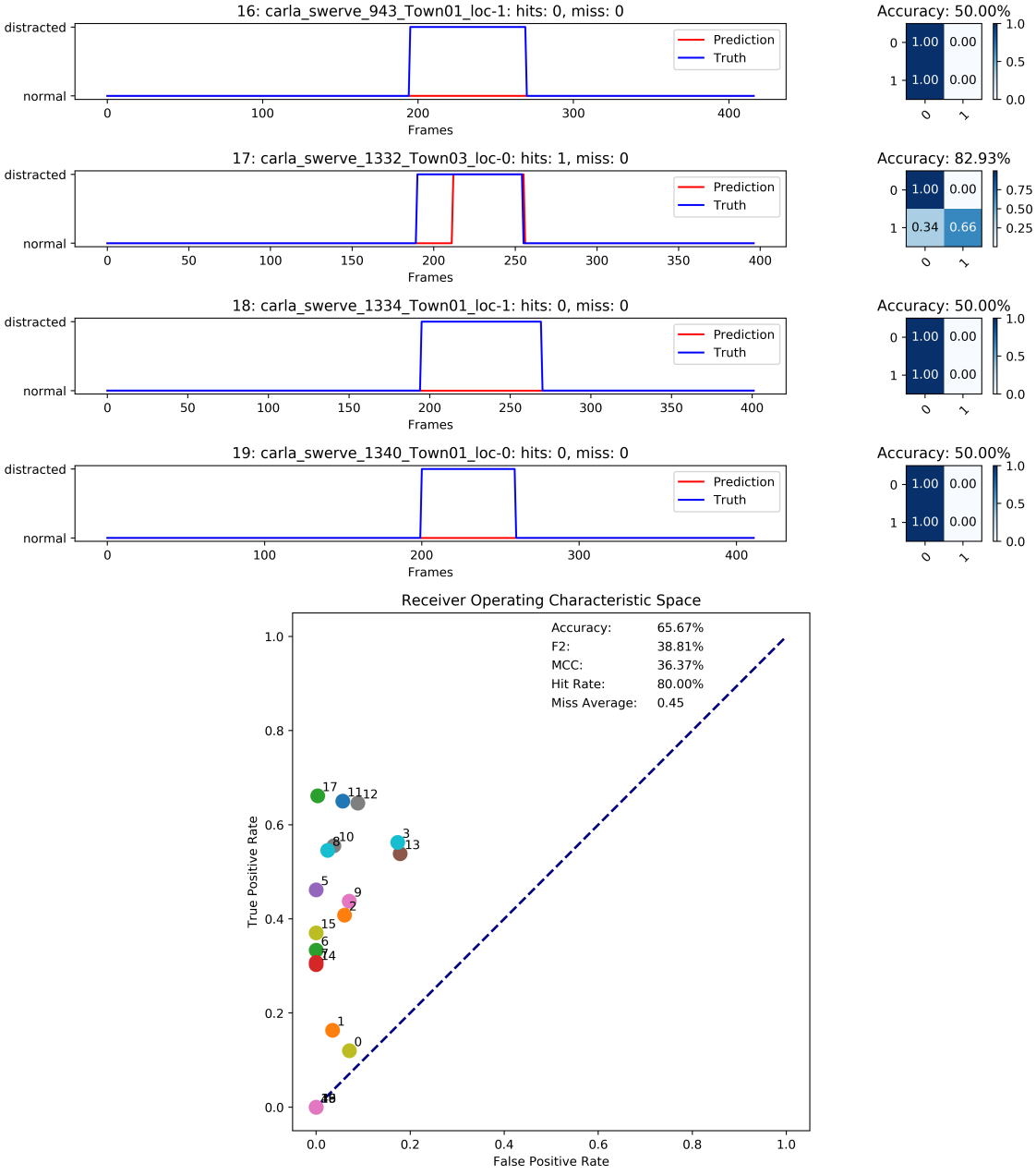



Figure A.12: CARLA Swerve Batch Part 3 of 3

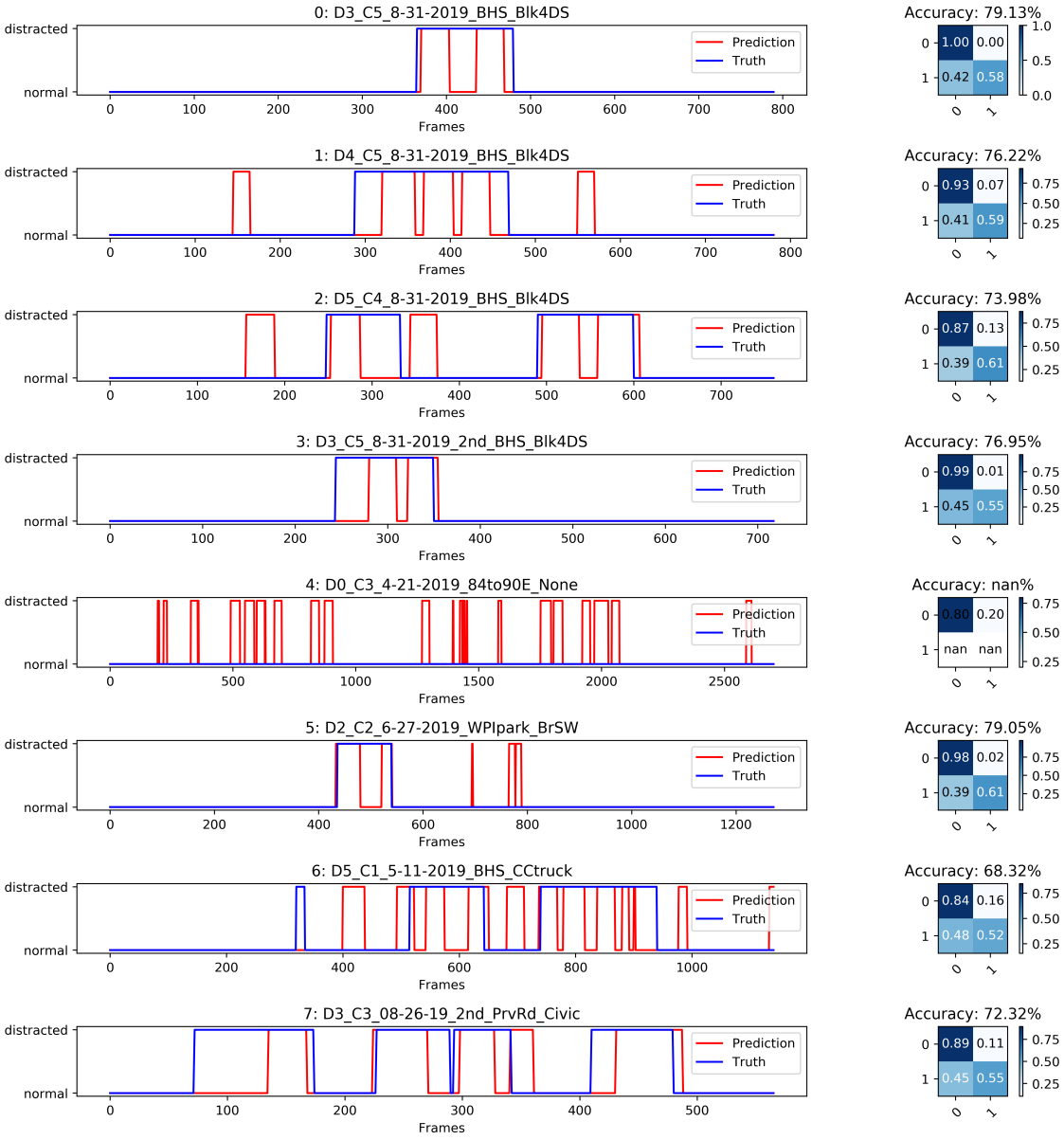



Figure A.13: Hardware Mix Batch Part 1 of 3

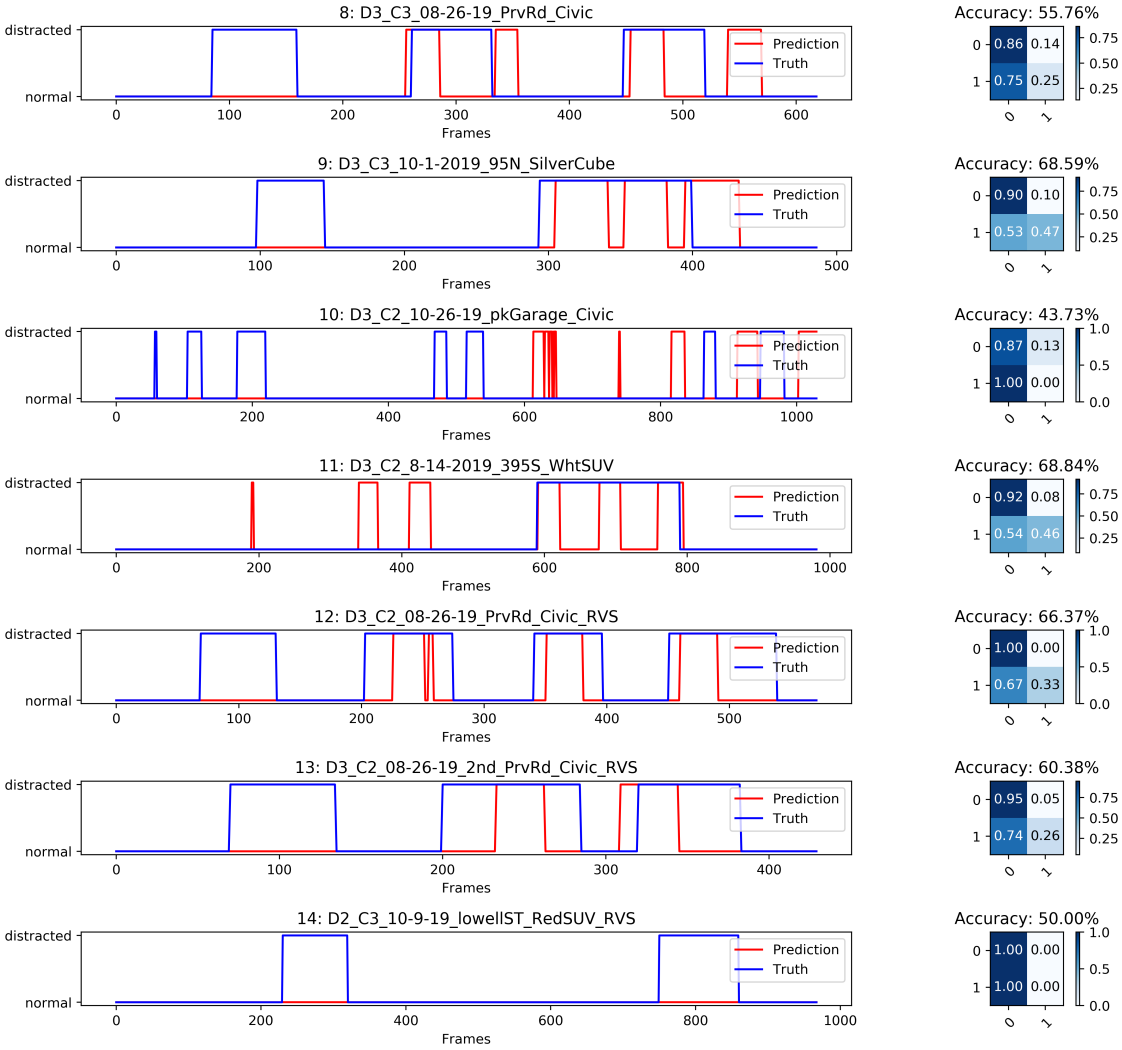



Figure A.14: Hardware Mix Batch Part 2 of 3

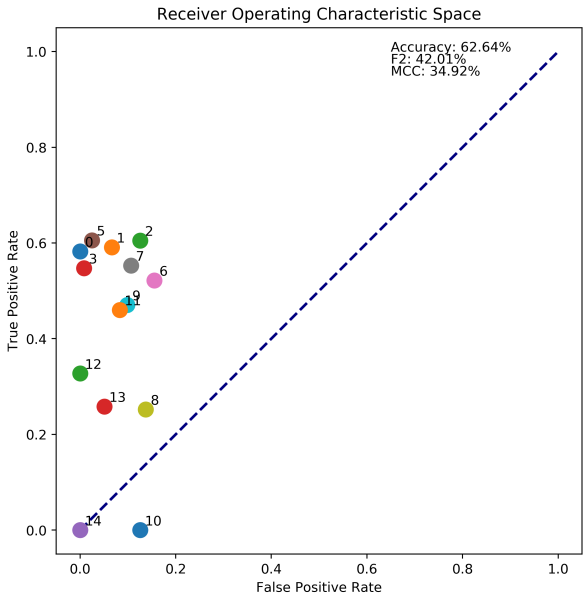



Figure A.15: Hardware Mix Batch Part 3 of 3

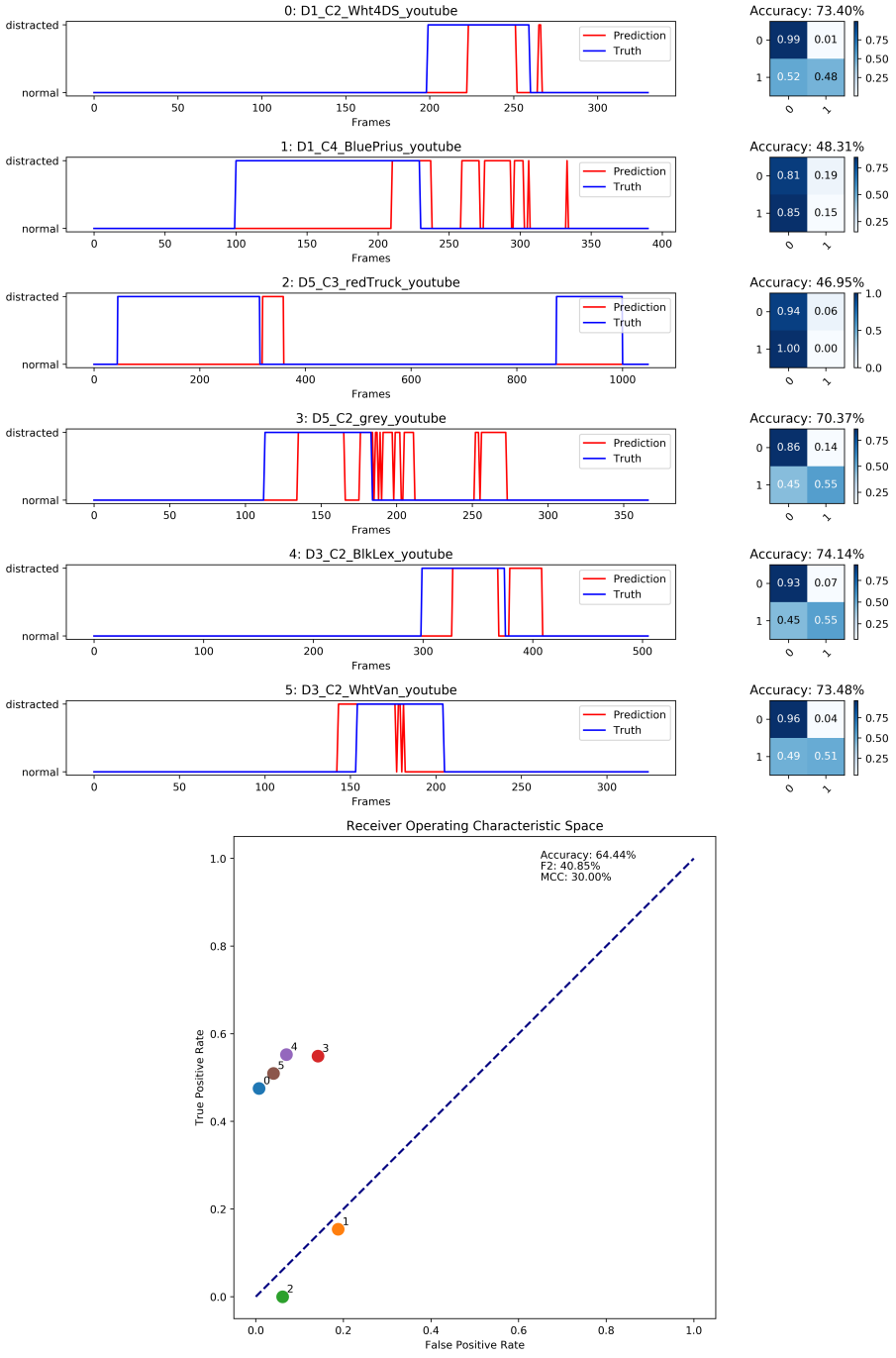



Figure A.16: YouTube Mix

# Appendix B

# ML Tests

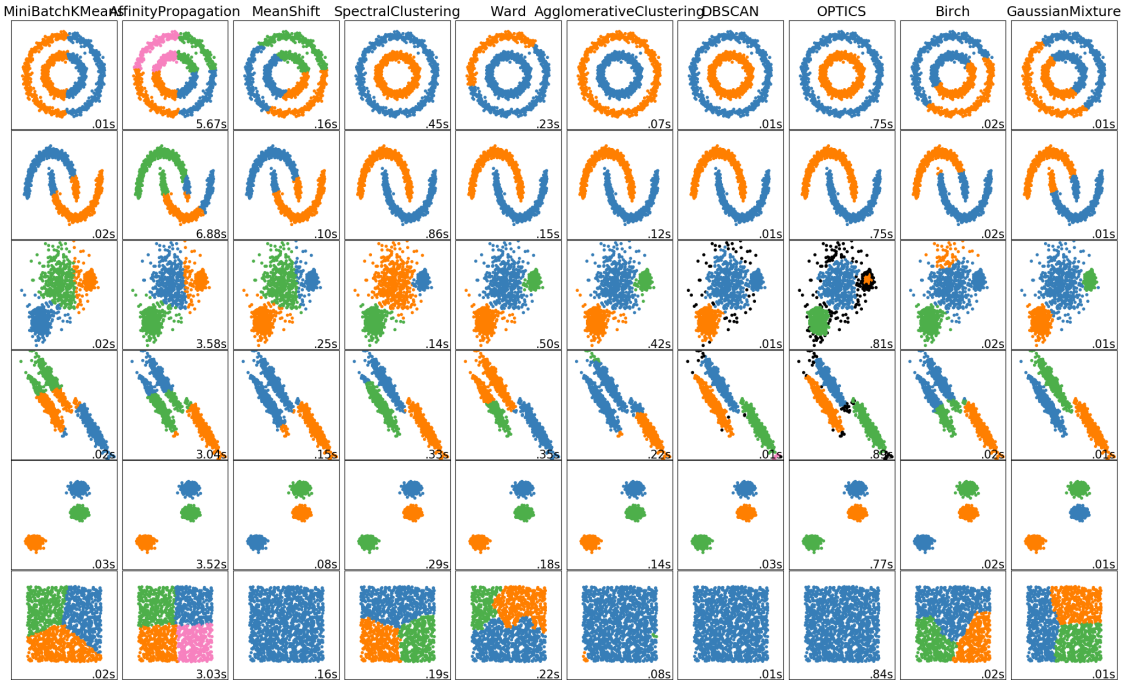



Figure B.1: Scikit Learn Clustering Example [78]

# Appendix C

# Source Code

## C.1 Python Test and Functions

```
#!/usr/bin/env python3
# default
import os
import sys
import time
import argparse
import datetime
import csv
import multiprocessing
import operator
import queue
import json
import math

# from pip
import yaml
import numpy as np
import pandas as pd
import cv2 as cv
import matplotlib.pyplot as plt
```

```
from termcolor import cprint
from pyfiglet import figlet_format
from moviepy.editor import VideoFileClip

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.utils.multiclass import unique_labels

# from this
from ObjectDetect import ObjectDetect, ObjectItem
from ObjectTrack import ObjectTrack
from AnomalyScore import AnomalyScore, ObjFeatures
from ClusterSeries import ClusterSeries
from BehaviorEst import BehaviorEst

#import pdb; pdb.set_trace() #DEBUG

class DDD():

    def __init__(self, parameters, is_live=False, replay=False):
        # target values
        self.img_width = 800
        self.img_height = 600

        # ddd settings
        self.tracker_limit = 2
        self.frame_counter = 0

        # debug settings
        self.det_en = True
        self.det_dbg = False
        self.track_en = True
        self.track_dbg = False
        self.behv_en = True
        self.tune = False
```

```
        # annotation settings
        self.font = cv.FONT_HERSHEY_SIMPLEX
        self.font_scale = 0.4
        self.font_thick = 1
        self.font_line = cv.LINE_AA

        self.marker = cv.MARKER_DIAMOND
        self.marker_size = 8
        self.marker_thick = 1

        self.main_color = (0, 255, 0) # green
        self.second_color = (66, 134, 244) # blue
        self.third_color = (102, 102, 153) # grey
        self.text_color = (255, 255, 153) # light yellow
        self.back_color = (0, 0, 0) # black
        self.laneL_color = (255, 153, 51) # orange
        self.laneR_color = (255, 102, 153) # pink
        self.mark_color = (0, 255, 0) # green
        self.warn_color = (255, 255, 0) # yellow
        self.dist_color = (255, 0, 0) # red

        # output
        self.output_image_fpt = ('./', 'result', '.jpg')

        # parameters
        anomaly_params = parameters['AnomalyScore']
        cluster_params = parameters['ClusterSeries']
        behav_params = parameters['BehaviorEst']

        # objects
        if(replay):
            # only import one track at a time during replay
            self.tracker_limit = 1

            self.detect_objects = None
            self.tracker = None
        else:
```

```
        self.detect_objects = ObjectDetect(self.img_width, self.img_height,
                                           min_thresh=0.35, max_det=40,
                                           debug=self.det_dbg)

        self.tracker = ObjectTrack(frames_till_forget=27,
                                           max_distance=50,
                                           tracker_limit=self.tracker_limit,
                                           min_track_corr=9,
                                           debug=self.track_dbg)

    self.features = ObjFeatures(max_track=self.tracker_limit,
                                is_live=is_live)

    self.anomaly = AnomalyScore(anomaly_params,
                                feature_list=self.features.feature_list,
                                max_track=self.tracker_limit,
                                is_live=is_live)

    self.cluster = []
    self.behav = []
    for i in range(self.tracker_limit):
        self.cluster.append(ClusterSeries(cluster_params, track=i))
        self.behav.append(BehaviorEst(behav_params, track=i))

def annotate_display(self, image, det_objs, trk_objs, related):
    """ Create annotations for the image based on objects and behavior
    """

    # combine lists if they exists
    objs = None
    if(trk_objs and det_objs):
        objs = det_objs + trk_objs
    elif(trk_objs):
        objs = trk_objs
    else:
        objs = det_objs
```

```
# run though all objects
if(objs):
    for obj in objs:
        anno_str = '{0}: {1:.2f}, {2}'.format(obj.label, obj.score, obj.uid)

        if(self.det_dbg):
            print('----------------------------------------')
            print(anno_str)

        # bounding box color
        if(obj.tracked and obj.focus):
            box_color = self.main_color
        elif(obj.tracked):
            box_color = self.second_color
        else:
            box_color = self.third_color

        # bounding box
        if(obj.tracked and (obj.distracted or obj.abnormal)):
            # double box
            offset = 5
            if(obj.distracted):
                box_color = self.dist_color
            elif(obj.abnormal):
                box_color = self.warn_color

            cv.rectangle(image,
                         (obj.bbox[0], obj.bbox[1]), (obj.bbox[2], obj.bbox[3]),
                         box_color, 2)

            cv.rectangle(image,
                         (obj.bbox[0] - offset, obj.bbox[1] - offset),
                         (obj.bbox[2] + offset, obj.bbox[3] + offset),
                         box_color, 2)
        else:
            cv.rectangle(image,
                         (obj.bbox[0], obj.bbox[1]), (obj.bbox[2], obj.bbox[3]),
```

```
                        box_color, 2)

            # box annotation back
            text_size, base = cv.getTextSize(anno_str, self.font,
                                             self.font_scale,
                                             self.font_thick)
            text_point = (obj.bbox[0], obj.bbox[1])
            text_off = (obj.bbox[0] + text_size[0],
                        obj.bbox[1] - text_size[1])
            cv.rectangle(image,
                        text_point,
                        text_off,
                        self.back_color, cv.FILLED)

            # box annotation
            cv.putText(image, anno_str, text_point,
                        self.font, self.font_scale,
                        self.text_color, self.font_thick,
                        self.font_line)

            # draw centroid if being tracked
            if(obj.tracked):
                marker_color = self.mark_color
                if(obj.distracted):
                    marker_color = self.dist_color
                elif(obj.abnormal):
                    marker_color = self.warn_color

                cv.drawMarker(image, tuple(obj.centroid), marker_color,
                              self.marker, self.marker_size,
                              self.marker_thick, self.font_line)

    else:
        if(self.det_dbg):
            print('No objects detected!')

    # add frame counter
```

```
        frame_info = 'Frame: {0}'.format(int(self.frame_counter))

        # outer box
        text_start = (5, 15)
        text_size, base = cv.getTextSize(frame_info, self.font,
                                         self.font_scale,
                                         self.font_thick)
        text_end = (text_start[0] + text_size[0],
                    text_start[1] - text_size[1])
        cv.rectangle(image,
                     text_start,
                     text_end,
                     self.back_color, cv.FILLED)
        # frame counter
        cv.putText(image, frame_info, text_start,
                     self.font, self.font_scale,
                     self.text_color, self.font_thick,
                     self.font_line)

        return image

    def contex_proc(self, objs):
        """ Attempt to understand which detection is the most important
        """

        # closest to the center along width only
        objs = sorted(objs, key=lambda x: x.centroid[0], reverse=True)

        return objs


    def image_pipeline(self, img, related=True):
        """ Run the full pipline of processing techniques on an image
        """

        rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        det_objs = None
```

```
trk_objs = None
beh_infer = {}

# get detections
if(self.det_en):
    det_objs = self.detect_objects.run(rgb)
    det_objs = self.contex_proc(det_objs)

# setup trackers
if(related and self.track_en):
    trk_objs = self.tracker.update_all(det_objs, rgb)

    if(self.track_dbg):
        print('frame: {0}, trackers: {1}'.format(self.frame_counter,
                      len(self.tracker.trackers)))

# understand behavior
if(self.behv_en):
    if(trk_objs):
        self.features.gen_features(trk_objs, self.frame_counter)

        # detect anomolous behavior
        self.anomaly.update(self.features, self.frame_counter, self.tune)

        for i in range(self.tracker.tracker_limit):
            # learn similar behavior
            self.cluster[i].update(self.frame_counter, self.features, self.anomaly.
                score)

            # infer behavior
            uid, beh = self.behav[i].update(self.cluster[i].hist_labels.array,
                                            self.cluster[i].hist_score.array,
                                            self.cluster[i].hist_size.array,
                                            self.anomaly.score.array[i, :],
                                            self.features.uid_key.array[i, :],
                                            self.frame_counter)
            if(uid is not None): beh_infer[uid] = beh
```

```
    # mark objects behavior
    if(trk_objs):
        for obj in trk_objs:
            # check uids
            if(obj.uid in beh_infer.keys()):
                if(beh_infer[obj.uid] == self.behav[0].state['abnormal']):
                    obj.abnormal = True
                    obj.distracted = False
                    continue

                if(beh_infer[obj.uid] == self.behav[0].state['distracted']):
                    obj.distracted = True
                    obj.abnormal = False
                    continue

            else:
                obj.abnormal = False
                obj.distracted = False

    # draw on image
    if(not self.det_dbg):
        det_objs = []

    img = self.annotate_display(img, det_objs, trk_objs,
                                related)
    if(related): self.frame_counter += 1
    return img


def run(self, image_file_path=None, video_file_path=None, debug=False, tune=False):
    # setup flags
    self.track_dbg = debug
    self.det_dbg = debug
    #
    self.tune = tune
```

```
# start text
cprint(figlet_format('start', font='cybermedium'),
       'green', 'on_grey', attrs=['bold'])

# find source type
if(image_file_path):
    # run on a set of images
    img = cv.imread(image_file_path)
    img = self.image_pipeline(img, False)

    # save output
    time_str = '{:%Y-%b-%d_%H:%M:%S:%f}'.format(datetime.datetime.now())
    output_fp = '{0}{1}_{2}{3}'.format(self.output_image_fpt[0],
                                       self.output_image_fpt[1],
                                       time_str, self.output_image_fpt[2])
    cv.imwrite(output_fp, img)

elif(video_file_path):
    # create results directory
    results_path = os.path.splitext(video_file_path)[0]

    # if it exists skip video
    if(os.path.exists(results_path)):
        print('NOTE: Video has an existing run in this location (remove if a re-run
            is needed)')
        return
    os.mkdir(results_path)

    # run on video file
    clip = VideoFileClip(video_file_path).\
                         fl_image(self.image_pipeline)
    clip.write_videofile('{0}/result.mp4'.format(results_path),
                         audio=False)

    # plot some features
    self.anomaly.plot(results_path, self.features,
```

```
                        self.frame_counter)
        self.anomaly.save(results_path, self.features,
                        self.frame_counter)

        for i in range(self.tracker.tracker_limit):
            self.cluster[i].plot(results_path, self.features,
                            self.frame_counter)
            self.behav[i].plot(results_path, self.features,
                            self.frame_counter)
    # end text
    cprint(figlet_format('stop', font='cybermedium'),
           'red', 'on_grey', attrs=['bold'])

def debug_run(self, prefix, pkl_filepath, truth_df=None, skip_plots=False):
    df = pd.read_pickle(pkl_filepath)
    ittr = len(df.index)

    # act on features as if running live
    for index, row in df.iterrows():
        self.features.update(row['uid'], 0, index,
                        row['area'], (row['pos_w'], row['pos_h']))


        # skip if there are no objects being tracked
        if(~np.isnan(row['uid'])):
            self.anomaly.update(self.features, index, False)

            # learn similar behavior
            self.cluster[0].update(index, self.features, self.anomaly.score)

            # infer behavior
            uid, beh = self.behav[0].update(self.cluster[0].hist_labels.array,
                                            self.cluster[0].hist_score.array,
                                            self.cluster[0].hist_size.array,
                                            self.anomaly.score.array[0, :],
                                            self.features.uid_key.array[0, :],
                                            index)
```

```
        # generate plots
        if(not skip_plots):
            self.anomaly.plot(prefix, self.features, ittr)
            self.cluster[0].plot(prefix, self.features, ittr)
            self.behav[0].plot(prefix, self.features, ittr)

        pred = None
        if(self.behav[0].get_pred(ittr) is not None):
            pred = self.behav[0].get_pred(ittr)

        # if we have truth generate confusion matrix
        cm = None
        if((truth_df is not None) and (pred is not None)):
            if(not skip_plots):
                cm = self.debug_plot(prefix, pred, truth_df)
            else:
                truth = truth_df['truth'].to_numpy()
                cm = confusion_matrix(truth, pred)

        return cm, pred

    def debug_plot(self, prefix, behav_pred, truth_df):
        truth = truth_df['truth'].to_numpy()

        # calculate truth
        _, cm = plot_confusion_matrix(truth, behav_pred, ['normal', 'abnormal'])
        plt.savefig('{0}/cm.png'.format(prefix), dpi=300)
        plt.close()

        return cm

# code from: https://scikit-learn.org/stable/auto_examples/model_selection/
    plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix
    -py
def plot_confusion_matrix(y_true, y_pred, classes,
                          normalize=True,
```

```
                title=None,
                cmap=plt.cm.Blues):
"""
This function prints and plots the confusion matrix.
Normalization can be applied by setting `normalize=True`.
"""
if not title:
    if normalize:
        title = 'Normalized confusion matrix'
    else:
        title = 'Confusion matrix, without normalization'

# Compute confusion matrix
cm = confusion_matrix(y_true, y_pred)
# Only use the labels that appear in the data
#classes = classes[unique_labels(y_true, y_pred)]
if normalize:
    cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       # ... and label them with the respective list entries
       xticklabels=classes, yticklabels=classes,
       title=title,
       ylabel='True label',
       xlabel='Predicted label')

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
```

```
    thresh = cm.max() / 2.
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            ax.text(j, i, format(cm[i, j], fmt),
                    ha="center", va="center",
                    color="white" if cm[i, j] > thresh else "black")
    fig.tight_layout()
    return ax, cm

def auto_batch_test(parameters, prefix, run_folder_path,
                        truth_file, skip_plots=True, verbose=False):

    # find folders which match standard video nameing format
    sub_folder_list = [x for x in os.listdir(run_folder_path)
                           if os.path.isdir(os.path.join(run_folder_path, x))]
    run_folder_list = [x for x in sub_folder_list
                           #if((x.startswith('D')) and ('_C' in x))]
                           if(not (x.startswith('_')))]

    # load in json
    with open(truth_file) as json_file:
        truth_labels = json.load(json_file)

    # loop through and run
    report_info = {}
    for dirs in run_folder_list:
        info = truth_labels[dirs]
        run_folder = os.path.join(run_folder_path, dirs)
        sub_prefix = os.path.join(prefix, dirs)
        os.mkdir(sub_prefix)

        # find associated pickle file from provided directory
        feat_fp = [x for x in os.listdir(run_folder)
                       if(x.startswith('trk') and x.endswith('feat.pkl'))]
        if(len(feat_fp) != 0):
            feat_fp.sort()
            feat_fp = os.path.join(run_folder, feat_fp[info['trk']])
```

```
else:
    feat_fp = None

# skip if track file is not found
if(feat_fp is None):
    print('NOTE: files not found for: {} - Track: {}'.format(dirs, feat_fp))
    continue

# load file and generate truth
df = pd.read_pickle(feat_fp)
frame_end = len(df.index)
truth = np.zeros(frame_end, dtype=np.int)

# mark truth array
if(info['dis'] is not None):
    for tup in info['dis']:
        truth[tup[0]:tup[1]] = 1
truth_df = pd.DataFrame({'truth': truth})

# init and run
if(verbose): print('Re-run on: {}\n Track: {}'.format(dirs, feat_fp))
ddd = DDD(parameters, is_live=False, replay=True)
cm, pred = ddd.debug_run(sub_prefix, feat_fp, truth_df, skip_plots)
if(verbose): print(cm)

# metric calculation
if((cm is not None)
        and (len(cm.ravel()) > 1)):

    tn, fp, fn, tp = cm.ravel()
    acc = (tp + tn) / (tp + tn + fp + fn)

    # F2 score
    beta = 2
    f2 = ((1 + beta**2)*tp) / ((1 + beta**2)*tp + (beta**2 * fn) + fp)

    # matthews correlation coefficient
```

```
            num = (tp*tn) - (fp*fn)
            den = math.sqrt((tp + fp)*(tp + fn)*(tn + fp)*(tn + fn))
            if(den == 0): den = 1
            mcc = num / den

            # true/false positive rates
            false_positive_rate = fp / (fp + tn)
            true_positive_rate = tp / (tp + fn)
        else:
            acc = 0
            f2 = 0
            mcc = -1
            false_positive_rate = 1
            true_positive_rate = 0


        # aggregate all confusion matricies for final report
        report_info[dirs] = {"cm": cm,
                             "truth": truth_df,
                             "pred": pred,
                             "order": info['odr'],
                             "tpr": true_positive_rate,
                             "fpr": false_positive_rate,
                             "accuracy": acc,
                             "F2": f2,
                             "MCC": mcc}

        # reset class for each loop
        ddd = None
    return report_info

def hit_and_miss_counter(pred, truth):
    # find truth falling and rising edges, should always be equal
    truth_redge = np.flatnonzero((truth[:-1] < 0.5) & (truth[1:] > 0.5))+1
    truth_fedge = np.flatnonzero((truth[:-1] > 0.5) & (truth[1:] < 0.5))+1

    if(truth_redge.shape == truth_fedge.shape):
```

```
    truth_hits = np.vstack((truth_redge, truth_fedge))
else:
    return np.nan, np.nan

# find predicted falling and rising edges, falling edge might be one short
pred_redge = np.flatnonzero((pred[:-1] < 0.5) & (pred[1:] > 0.5))+1
pred_fedge = np.flatnonzero((pred[:-1] > 0.5) & (pred[1:] < 0.5))+1

if(pred_redge.shape == pred_fedge.shape):
    pred_hits = np.vstack((pred_redge, pred_fedge))

elif((pred_fedge.shape[0] + 1) == pred_redge.shape[0]):
    pred_fedge = np.append(pred_fedge, pred.shape[0])
    pred_hits = np.vstack((pred_redge, pred_fedge))

else:
    return np.nan, np.nan

# loop through truth hits and count hits and remove from pred hits array
hit_count = 0
miss_count = 0
for t_col in truth_hits.T:
    detected = False
    hit_idx = []

    for i, p_col in enumerate(pred_hits.T):
        # does the positive predition occur when truth is positive
        if((t_col[0] <= p_col[0] <= t_col[1])
               or (t_col[0] <= p_col[1] <= t_col[1])):

            detected = True
            hit_idx.append(i)

    if(detected):
        hit_count += 1
        pred_hits = np.delete(pred_hits, hit_idx, 1)
```

```
    miss_count = pred_hits.shape[1]

    return hit_count, miss_count

def generate_final_report(prefix, report_info):

        report_info = dict(sorted(report_info.items(),
                              key=lambda x:operator.getitem(x[1], 'order')))
        ###
        # FIGURE 0 - one large figure
        ###
        run_count = len(report_info.keys())
        figure_size_tuple = (run_count+4, 4)
        total_hit = 0
        total_miss = []

        # itterate through runs
        fig = plt.figure(num=0, figsize=(12, 2*run_count))
        for i, (key, value) in enumerate(report_info.items()):
            truth = value['truth']['truth'].to_numpy()
            pred = value['pred']
            cm = value['cm']
            row = value['order']
            hit_cnt, miss_cnt = hit_and_miss_counter(pred, truth)

            total_hit += np.nan_to_num(hit_cnt)
            total_miss.append(miss_cnt)

            # truth vs pred
            axs = plt.subplot2grid(figure_size_tuple, (row, 0), colspan=3)
            if(pred is not None): axs.plot(pred, color='red', label='Prediction')
            axs.plot(truth, color='blue', label='Truth')
            axs.set_title('{}: {}: hits: {}, miss: {}'.format(row, key, hit_cnt, miss_cnt))
            axs.set_xlabel('Frames')
            axs.set_ylim([-0.05, 1.05])
            axs.set_yticks(np.arange(2))
            axs.set_yticklabels(['normal', 'distracted'])
```

```
        axs.legend(loc='upper right')

        # cm and metrics
        axs = plt.subplot2grid(figure_size_tuple, (row, 3), colspan=1)
        if(cm is not None):
            im = axs.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
            axs.figure.colorbar(im, ax=axs)
            axs.set_title('Accuracy: {:2.2f}%'.format(100 * value['accuracy']))

            # Rotate the tick labels and set their alignment.
            plt.setp(axs.get_xticklabels(), rotation=45, ha="right",
                     rotation_mode="anchor")

            # Loop over data dimensions and create text annotations.
            fmt = '.2f'
            thresh = cm.max() / 2.
            for i in range(cm.shape[0]):
                for j in range(cm.shape[1]):
                    axs.text(j, i, format(cm[i, j], fmt),
                             ha="center", va="center",
                             color="white" if cm[i, j] > thresh else "black")

    # ROC plot with a point per run
    axs = plt.subplot2grid(figure_size_tuple, (run_count, 0), colspan=4, rowspan=4)
    axs.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    # add a point per run
    for key, value in report_info.items():
        if(pred is not None):
            axs.plot(value['fpr'], value['tpr'],
                     marker='o', markersize=11)
            text_offset = 0.01
            axs.annotate(str(value['order']),
                     (value['fpr']+ text_offset,
                     value['tpr']+ text_offset))

    # format
    total_acc = np.mean(np.nan_to_num(np.asarray(
```

```
                    [v['accuracy'] for k,v in report_info.items()]
                )))
total_mcc = np.mean(np.nan_to_num(np.asarray(
                    [v['MCC'] for k,v in report_info.items()]
                )))
total_f2 = np.mean(np.nan_to_num(np.asarray(
                    [v['F2'] for k,v in report_info.items()]
                )))

start_y = 1.01
dec_y = 0.04
start_x = 0.50
nstart_x = 0.725
axs.text(start_x, start_y,
           'Accuracy:',
           color='black')
axs.text(nstart_x, start_y,
           '{:2.2f}%'.format(100 * total_acc),
           color='black')

axs.text(start_x, start_y - (1*dec_y),
           'F2:',
           color='black')
axs.text(nstart_x, start_y - (1*dec_y),
           '{:2.2f}%'.format(100 * total_f2),
           color='black')

axs.text(start_x, start_y - (2*dec_y),
           'MCC:',
           color='black')
axs.text(nstart_x, start_y - (2*dec_y),
           '{:2.2f}%'.format(100 * total_mcc),
           color='black')

axs.text(start_x, start_y - (3*dec_y),
           'Hit Rate:',
           color='black')
```

```
        axs.text(nstart_x, start_y - (3*dec_y),
                    '{:2.2f}%'.format(100 * (total_hit / run_count)),
                    color='black')

        axs.text(start_x, start_y - (4*dec_y),
                    'Miss Average:',
                    color='black')
        axs.text(nstart_x, start_y - (4*dec_y),
                    '{:2.2f}'.format(np.mean(np.nan_to_num(total_miss))),
                    color='black')

        axs.set_aspect('equal')
        axs.set_title('Receiver Operating Characteristic Space')
        axs.set_xlabel('False Positive Rate')
        axs.set_ylabel('True Positive Rate')

        # save
        fig.tight_layout()
        plt.savefig('{0}/final_report.png'.format(prefix), dpi=400)
        plt.close()

def parse_arguments():
    # parse command line arguments
    parser = argparse.ArgumentParser()

    # arguments
    parser.add_argument('-i', '--image',
                        help='Run on a single image (provide path)')

    parser.add_argument('-v', '--video',
                        help='Run on a single video (provide path)')

    parser.add_argument('-p', '--parameters',
                        default='./parameters.yaml',
                        help='YAML file used to store parameters (provide path)')

    parser.add_argument('-d', '--debug', action='store_true',
```

```
                        help='Print extra debug information')

    parser.add_argument('-t', '--tune', action='store_true',
                        help='Pause playback for algorithm plotting')

    parser.add_argument('-pp', '--pickle_playback',
                        help='Pickle file used to replay previous run (provide path)')

    parser.add_argument('-pt', '--pickle_truth',
                        help='Pickle file used for truth comparison (provide path)')


    parser.add_argument('-sp', '--skip_plots', action='store_true',
                        help='Skip individual run plot generation')

    parser.add_argument('-bt', '--batch_test',
                        help='Loop over a folder consisting of videos and run each\
                                one (provide path)')

    parser.add_argument('-abt', '--auto_batch_test',
                        help='Loop over pickle files found for previously\
                                ran videos and search for existing truth\
                                information for comparison (provide\
                                path to completed run folders and path\
                                to truth files with -tf)')

    parser.add_argument('-tf', '--truth_file',
                        default='../data/truth/labels.json',
                        help='JSON file containing truth label information(provide path)')

    return parser.parse_args()

def main():
    args = parse_arguments()

    # welcome text
    cprint(figlet_format('DDD !!!', font='starwars'),
```

```
        'yellow', 'on_grey', attrs=['bold'])

# load parameters
with open(args.parameters) as f:
    parameters = yaml.load(f, Loader=yaml.FullLoader)

if(args.batch_test):
    # run for all files in provided folder
    for filename in next(os.walk(args.batch_test))[2]:

        # find images
        if(filename.endswith('.png') and not ('.mp4' in filename)):
            print(filename)

            ddd = DDD(parameters)
            ddd.run(os.path.join(args.batch_test, filename), None, args.debug, args.tune)

        # find videos
        if((filename.endswith('.mp4') or filename.endswith('.m4v')) and
                not ("result" in filename)):
            print('Video File: {}'.format(filename))

            ddd = DDD(parameters)
            ddd.run(None, os.path.join(args.batch_test, filename), args.debug, args.tune)

        # reset class for each loop
        ddd = None

elif(args.auto_batch_test):
    if(args.truth_file is None):
        print('NOTE: please specify truth filepath with -tf')
        return

    # rerun text
    cprint(figlet_format('Re-Run BATCH!', font='cybermedium'),
            'blue', 'on_grey', attrs=['bold'])
```

```
    # create a new directory to host all re-runs
    og_prefix = './batch_rerun'
    prefix = og_prefix
    i = 0
    while(os.path.exists(prefix)):
        prefix = og_prefix + '_{0}'.format(i)
        i = i + 1
    os.mkdir(prefix)

    # run auto batch
    report_info = auto_batch_test(parameters, prefix,
                                  args.auto_batch_test, args.truth_file,
                                  args.skip_plots, True)

    # save report
    generate_final_report(prefix, report_info)


elif(args.pickle_playback):
    # rerun text
    cprint(figlet_format('Re-Run!', font='cybermedium'),
           'blue', 'on_grey', attrs=['bold'])

    # create a new folder for the re-run
    og_prefix = './rerun_ddd'
    prefix = og_prefix
    i = 0
    while(os.path.exists(prefix)):
        prefix = og_prefix + '_{0}'.format(i)
        i = i + 1
    os.mkdir(prefix)

    # load truth data frame
    truth_df = None
    if(args.pickle_truth):
        truth_df = pd.read_pickle(args.pickle_truth)
```

```
        # init and run
        ddd = DDD(parameters, is_live=False, replay=True)
        cm, _ = ddd.debug_run(prefix, args.pickle_playback, truth_df)
        print(cm)

    else:
        # init and run
        ddd = DDD(parameters)
        ddd.run(args.image, args.video, args.debug, args.tune)



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

```
#!/usr/bin/env python3
'''
    Anomaly score with signal processing
'''
#default
import copy
import os
import argparse

# from pip
import numpy as np
import pandas as pd

# only supports saving to a file
# otherwise we get GTK conflicts with openCV
import matplotlib
matplotlib.use('AGG')
import matplotlib.pyplot as plt

# from this
```

```
from Zscore import Zscore
from ObjectFeatures import ObjFeatures, HistArray

class AnomalyScore():
    def __init__(self, params, feature_list, max_track, is_live=False, simple=False):
        self.is_live = is_live
        self.simple = simple

        # z-score
        self.z_score = {}
        self.z_score['norm'] = Zscore(lag = params['zscore']['norm']['lag'],
                              threshold = params['zscore']['norm']['threshold'],
                              influence = params['zscore']['norm']['influence'],
                              memory_size = max_track * len(feature_list))

        self.z_score['diff'] = Zscore(lag = params['zscore']['diff']['lag'],
                              threshold = params['zscore']['diff']['threshold'],
                              influence = params['zscore']['diff']['influence'],
                              memory_size = max_track * len(feature_list))
        # simple
        self.s_window = 60
        self.s_std = 1.8

        # FFT
        self.fft_win = params['fft']['window']
        self.fft_bins = 20
        self.fft_Fs = 1
        self.fft_range = 0.2
        self.fft_nan_max = 0.4
        self.fft_lim = 10

        # distraction scoring
        self.score = HistArray(rows=max_track,
                               dtype=np.uint8)
        self.score_feat = params['score']

        # gate anything that adds a score with a min peak time
```

```
        self.anom_min_frame = params['peak']['min_frame']
        self.peak_counter = np.zeros(shape=(max_track, len(self.score_feat.keys())))

        # plotting help
        self.peaks= {}
        for fkey in feature_list:
            self.peaks[fkey] = HistArray(rows=max_track,
                                         keep=(not self.is_live))

    def sig_detect(self, features, fcnt, tune=False):
        """ Process signal and save off score
        """
        score_total = np.zeros(features.max_track)

        # look through all smooth features
        for i, (key, ft) in enumerate(features.sfeat.items()):
            if(ft.array is not None):
                # rotate through track
                for trk_idx in range(ft.array.shape[0]):

                    #TODO find a way to integrate this into the normal score
                    # this is sub-sampled attempting to find jumps in trends
                    if(self.simple and fcnt > self.s_window):

                        # anomaly detection
                        s = ft.array[trk_idx, fcnt-self.s_window:fcnt]

                        # resample
                        s = s[::10]

                        mean = np.mean(s)
                        std = np.std(s)

                        over = s[-1] > (mean + self.s_std*std)
                        under = s[-1] < (mean - self.s_std*std)

                        if(over or under):
```

```
                self.peaks[key].update(trk_idx, fcnt, 1)
                score_total[trk_idx] = score_total[trk_idx] + self.score_feat[key]

            # oscillation detection
            # find FFT; only for pos_w
            if(key == "pos_w"):
                s = ft.array[trk_idx, :]
                hist = self.an_fft(fcnt, s)

                # score is a fraction of the displaced energy
                if(hist is not None):
                    score_total[trk_idx] = score_total[trk_idx] + \
                        ((np.sum(hist[2:5]) / np.sum(hist)) * \
                            self.score_feat['pos_w_fft'])

        else:
            # index calc for z_score
            z_index = i
            if(ft.array.shape[0] > 1):
                z_index = (i + (trk_idx + i))

            # find peaks
            if(fcnt > np.max([self.z_score['norm'].lag, self.z_score['diff'].lag])
                ):
                found = None
                s = None

                if(key == "pos_w"):
                    s = ft.array[trk_idx, fcnt-self.z_score['norm'].lag:fcnt]
                    found = self.z_score['norm'].run(s, z_index)

                elif((key == "pos_h") or (key == "area")):
                    s = features.dfeat[key].array[trk_idx, fcnt-self.z_score['diff
                        '].lag:fcnt]
                    found = self.z_score['diff'].run(s, z_index)

                if(found is not None):
```

```
                self.peak_counter[trk_idx, i] = self.peak_counter[trk_idx, i]
                    + 1
                if(self.peak_counter[trk_idx, i] >= self.anom_min_frame):
                    self.peaks[key].update(trk_idx, fcnt, 1)
                    score_total[trk_idx] = score_total[trk_idx] + self.
                        score_feat[key]

            else:
                self.peak_counter[trk_idx, i] = 0


            # print tunning plots
            if(tune and
               (s is not None) and
               (fcnt > self.fft_win) and
               (fcnt % (self.fft_win // 4) == 0)):

                title = '(f: {3}, key: {0}, trk: {1}) = {2}'.format(key,
                    trk_idx, found, fcnt)

                if(key == "pos_w"):
                    self.z_score['norm'].plot(title, s, z_index)

                elif((key == "pos_h") or (key == "area")):
                    self.z_score['diff'].plot(title, s, z_index)

        # find FFT, only for pos_w
        if(key == "pos_w"):
            # only run fft every windown/4 frame
            if((fcnt > self.fft_win) and
                (fcnt % (self.fft_win // 2) == 0)):

                s = ft.array[trk_idx, :]
                hist = self.an_fft(fcnt, s, tune)

                # score is based on fft small bin ratio
                if(hist is not None):
```

```
                                fft_score = (np.sum(hist[1:4]) / np.sum(hist)) * \
                                       self.score_feat['{0}_fft'.format(key)]

                                score_total[trk_idx] = score_total[trk_idx] + fft_score

                                # allow the score to impact past values
                                score_window = (fcnt - (self.fft_win // 2))
                                self.score.array[trk_idx, score_window:(fcnt-1)] =
                                    fft_score + \
                                     self.score.array[trk_idx, score_window:(fcnt-1)]


        # add up score and flag uids
        if(features.uid_key.array is not None):
            for i, score in enumerate(score_total):
                self.score.update(i, fcnt, score)

        return True

    def an_fft(self, fcnt, signal, tune=False):
        s = signal[fcnt-self.fft_win:fcnt]

        # if we have over (nan_max)% NaNs don't run FFT
        mask = np.isfinite(s)
        if((np.sum(mask) / self.fft_win) > self.fft_nan_max):

            # linear interpolate NaNs
            si = np.arange(len(s))
            s_interp = np.interp(si, si[mask], s[mask])

            # shift mean to 0
            s_interp = s_interp - np.mean(s_interp)

            # FFT
            fft = np.fft.hfft(s_interp)

            # get positive half and normalize by sample amount
```

```
        hist, bin_edges = np.histogram(np.abs(fft)[:self.fft_win // 2] * 1 / self.
            fft_win,
                                       bins=self.fft_bins, range=(0,self.fft_range))

        # print tunning plots
        if(tune):
            print(hist)
            print(bin_edges)
            fig, axs = plt.subplots(2, 1)
            axs[0].set_ylabel("Amplitude")
            axs[0].set_xlabel("Frequency [Hz]")
            N = self.fft_win
            f = np.linspace(0, self.fft_Fs, self.fft_win)
            axs[0].bar(bin_edges[:-1], hist, color='c', edgecolor='b',
                       width=self.fft_range/self.fft_bins)
            axs[0].axhline(y=self.fft_lim, color='r', linestyle='--')
            axs[0].set_ylim([0, 70])

            axs[1].set_xlabel("Frames")
            axs[1].set_ylabel("Position [pixels]")
            axs[1].set_ylim([-100, 100])
            axs[1].plot(s_interp)
            fig.tight_layout()
            plt.savefig('fft_tune.png', dpi=300)
            input("Press Enter...")
            plt.show()

        return hist



#TODO might want to remove this
def update(self, features, fcnt, tune=False):
    self.sig_detect(features, fcnt, tune)


def plot(self, prefix, features, frame_end):
```

```
###
# FIGURE 0
###
if(self.score.array is not None):
    # score
    trk_amt = self.score.array.shape[0]
    fig, axs = plt.subplots(trk_amt, 1)
    max_score = np.sum(list(self.score_feat.values()))

    if(trk_amt > 1):
        for i in range(trk_amt):
            y = self.score.array[i, :frame_end]
            axs[i].plot(y, color='b', label='score')
            #
            axs[i].set_ylim([0, max_score])
            axs[i].set_xlim([0, frame_end])
            axs[i].set_ylabel('Score')
            axs[i].set_xlabel('Frames')
            #
            major_ticks = np.arange(0, y.shape[0]+1, 100)
            minor_ticks = np.arange(0, y.shape[0]+1, 20)
            axs[i].set_xticks(major_ticks)
            axs[i].set_xticks(minor_ticks, minor=True)
            major_ticks = np.arange(0, max_score+1, 1)
            minor_ticks = np.arange(0, max_score+1, 0.2)
            axs[i].set_yticks(major_ticks)
            axs[i].set_yticks(minor_ticks, minor=True)
            axs[i].grid(which='minor', alpha=0.2)
            axs[i].grid(which='major', alpha=0.5)
            #
            axs[i].legend(loc="upper right")
    else:
        y = self.score.array[0, :frame_end]
        axs.plot(y, color='b', label='score')
        axs.set_ylim([0, max_score])
        axs.set_xlim([0, frame_end])
```

```
            #
            major_ticks = np.arange(0, y.shape[0]+1, 100)
            minor_ticks = np.arange(0, y.shape[0]+1, 20)
            axs.set_xticks(major_ticks)
            axs.set_xticks(minor_ticks, minor=True)
            major_ticks = np.arange(0, max_score+1, 1)
            minor_ticks = np.arange(0, max_score+1, 0.2)
            axs.set_yticks(major_ticks)
            axs.set_yticks(minor_ticks, minor=True)
            axs.grid(which='minor', alpha=0.2)
            axs.grid(which='major', alpha=0.5)
            #
            axs.set_ylabel('Score')
            axs.set_xlabel('Frames')
            axs.legend(loc="upper right")

        fig.tight_layout()
        plt.savefig('{0}/score.png'.format(prefix), dpi=300)
        plt.close()

    # only plot score if live
    # otherwise post processes for other plots
    if(self.is_live):
        return

    # signal and peak information
    for key, ft in features.feat.items():
        if(ft.array is None):
            continue
        trk_amt = ft.array.shape[0]

        ###
        # FIGURE 1
        ###
        for i in range(trk_amt):
            fig, axs = plt.subplots(2, 1)
```

```
# plot raw and smooth data
# shift raw to match smooth delay
raw = np.append(np.zeros(features.smooth_win // 2), ft.array[i,:frame_end])
axs[0].plot(raw, c='b', label='raw',linewidth=1.0)
axs[0].plot(features.sfeat[key].array[i,:frame_end], c='r',
              label='smooth',linewidth=1.0)

# plot peak information
if(self.peaks[key].array is not None):
    peaks = self.peaks[key].array[i,:frame_end]
    y_values = features.sfeat[key].array[i,:frame_end] * peaks
    axs[0].plot(y_values, marker='+', markersize=5, c='g', label='peaks')

# settings
if(key == "area"):
    axs[0].set_ylabel('Position [pixels$^2$]'.format(key))
else:
    axs[0].set_ylabel('Position [pixels]'.format(key))
axs[0].set_xlabel('Frames')
axs[0].set_xlim([0, frame_end])
axs[0].grid(True)
axs[0].legend(loc="upper right")

# diff data
axs[1].plot(features.dfeat[key].array[i,:frame_end], c='c',
              label='d/df',linewidth=1.0)

# peak diff data
if(self.peaks[key].array is not None):
    peaks = self.peaks[key].array[i,:frame_end]
    y_values = features.dfeat[key].array[i,:frame_end] * peaks
    axs[1].plot(y_values, marker='+', markersize=5, c='g', label='peaks')

# settings
if(key == "area"):
    axs[1].set_ylabel('Difference [pixels$^2$]'.format(key))
else:
```

```
                axs[1].set_ylabel('Difference [pixels]'.format(key))
            axs[1].set_xlabel('Frames')
            axs[1].set_xlim([0, frame_end])
            axs[1].grid(True)
            axs[1].legend(loc="upper right")

            fig.tight_layout()
            plt.savefig('{1}/{2}_trk_{0}_val.png'.format(key, prefix, i), dpi=500)
            plt.close()

'''
    Debug and extra functions
'''
def tune_plot(self, df):
    sig = df.loc[:, 'pos_w'].values
    means = np.zeros_like(sig)
    stds = np.zeros_like(sig)

    # loop
    for i in range(sig.shape[0]):
        if(i > self.s_window):
            s = sig[i-self.s_window:i]
            means[i] = np.mean(s)
            stds[i] = np.std(s)

    # plot
    plt.figure(1)
    plt.title('mean and std')
    plt.plot(sig, c='g', label='signal')
    plt.plot(means, c='b', label='mean')

    plt.plot(means + (self.s_std * stds), c='r', label='upper')
    plt.plot(means - (self.s_std * stds), c='r', label='lower')
    plt.legend()
    plt.savefig('./tune.png', dpi=500)
    plt.close()
```

```
def debug_run(self, pkl_filepath, tune_only=False):
    df = pd.read_pickle(pkl_filepath)
    ittr = len(df.index)

    if(tune_only):
        self.tune_plot(df)
        return

    # only import one track at a time
    features = ObjFeatures(max_track=1,
                           is_live=False,
                           buffer_size=ittr)

    # act on features as if running live
    for index, row in df.iterrows():
        features.update(row['uid'], 0, index,
                        row['area'], (row['pos_w'], row['pos_h']))

        # run score
        self.sig_detect(features, index)


    # save off features with updated new score
    prefix_og = './rerun_anom'
    prefix = prefix_og
    i = 0
    while(os.path.exists(prefix)):
        prefix = prefix_og + '_{0}'.format(i)
        i = i + 1

    os.mkdir(prefix)
    self.plot(prefix, features, ittr)
    self.save(prefix, features, ittr)


def save(self, prefix, features, frame_end):
```

```
        # convert all numpy arrays to pandas data frame per track
        for i in range(features.max_track):
            # setup initial dataframe info
            df = pd.DataFrame({'uid': features.uid_key.array[i, :frame_end],
                               'score': self.score.array[i, :frame_end]})

            # add features dynamically
            for fkey in features.feature_list:
                df[fkey] = features.feat[fkey].array[i, :frame_end]

            # save
            df.to_pickle('{0}/trk{1}_feat.pkl'.format(prefix, i))

def main():
    # parse command line arguments
    parser = argparse.ArgumentParser()

    # arguments
    parser.add_argument('-p', '--rerun',
                        help='Load feature data frame (provide path)')

    args = parser.parse_args()

    # setup objects
    features = ObjFeatures(max_track=1) #TODO parameters
    anomaly = AnomalyScore(None, feature_list=features.feature_list,
                                            max_track=1)
    # run
    if(args.rerun is not None):
        anomaly.debug_run(args.rerun)


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

```
'''
    Behavior estimation from
    signal processing and machine learning
'''
# default
import copy
import random

# from pip
import numpy as np
import pandas as pd

# only supports saving to a file
# otherwise we get GTK conflicts with openCV
import matplotlib
matplotlib.use('AGG')
import matplotlib.pyplot as plt

# from this
from ObjectFeatures import ObjFeatures, HistArray

class BehaviorEst():
    """ This class makes an inference about the drivers behavior based on the other
        processing blocks
    """

    def __init__(self, params, track=0, debug=False):
        self.state = {'normal' : 0,
                      'abnormal' : 1,
                      'distracted' : 2
                      }
        self.current_state = self.state['normal']
        self.next_state = self.state['normal']
        self.trk = track
        self.debug = debug

        # wait time before state machine starts
```

```
self.min_hist_frame = 30
self.state_counter = 0

# state machine settings
self.state_trans_frame = np.zeros(shape=(len(self.state.keys()), len(self.state.keys
    ())),
                                  dtype=np.dtype(int))
# current state, next state = min frames for a transition
self.state_trans_frame[self.state['normal'], self.state['abnormal']] = 10
self.state_trans_frame[self.state['abnormal'], self.state['normal']] = 20
self.state_trans_frame[self.state['abnormal'], self.state['distracted']] = 10
self.state_trans_frame[self.state['distracted'], self.state['normal']] = 20

# anomaly settings
anom_conditionals = 2
self.anom_win = np.zeros(shape=(len(self.state.keys()), anom_conditionals),
                                  dtype=np.dtype(int))

# 0 - up, 1 - down
spar = params['anomaly']['window']
self.anom_win[self.state['normal'], 0] = spar['normal_up']
self.anom_win[self.state['abnormal'], 0] = spar['abnormal_up']
self.anom_win[self.state['abnormal'], 1] = spar['abnormal_down']
self.anom_win[self.state['distracted'], 1] = spar['distracted_down']

self.anom_score_thresh = params['anomaly']['score_threshold']

# cluster settings
lrn_conditionals = 2
self.lrn_win = np.zeros(shape=(len(self.state.keys()), lrn_conditionals),
                                  dtype=np.dtype(int))

spar = params['learn']['window']
self.lrn_win[self.state['normal'], 0] = spar['normal_up']
self.lrn_win[self.state['abnormal'], 0] = spar['abnormal_up']
self.lrn_win[self.state['abnormal'], 1] = spar['abnormal_down']
self.lrn_win[self.state['distracted'], 0] = spar['distracted_stay']
```

```
        self.lrn_win[self.state['distracted'], 1] = spar['distracted_down']

        self.lrn_score_min = self.anom_score_thresh * params['learn']['score_threshold_ratio
            ']

        # conditional probabilities
        self.anom_cond = np.zeros(shape=self.anom_win.shape)
        self.lrn_cond = np.zeros(shape=self.lrn_win.shape)

        spar = params['learn']['probability']
        self.lrn_cond[self.state['normal'], 0] = spar['normal_up']
        self.lrn_cond[self.state['distracted'], 1] = spar['distracted_stay']
        self.lrn_cond[self.state['distracted'], 0] = spar['distracted_down']

        # history
        self.hist_infer = HistArray(rows=1, dtype=np.uint16)


    def update(self, clu_label, clu_score, clu_size, anom_score, anom_uid, fcnt):
        """ Evaluates, per frame, the anomolous behavior of the driver

            Arguments
                clu_label [ndarray (1, frames)] : label of the current frame
                clu_score [ndarray (cluster_amt, frames)] : score of every cluster
                clu_size [ndarray (cluster_amt, frames)] : size of every cluster
                anom_score [ndarray (1, frames)] : score from anomaly detector
                anom_uid [ndarray (1, frames)] : frames current UID

            Returns
               uid [int]: UID of object under evaluation, None if nothing to infer
               infer [int]: objects behavior inference
        """
        # must have a min history amount and valid arrays
        if((fcnt < self.min_hist_frame)
               or (clu_score is None)
               or (clu_label is None)
               or (anom_score is None)):
```

```
        return None, None


    # do we have a label for this frame
    if(not np.isnan(clu_label[0, fcnt])):
        # get current label as index
        current_label = int(clu_label[0, fcnt])

        # order clusters by score
        label_byscore_asc = clu_score[:,fcnt].argsort()
    else:
        current_label = None

    # roll the dice
    roll = random.uniform(0, 1)

    # helpers
    score_up = anom_score[(fcnt - self.anom_win[self.current_state, 0]): fcnt]
    score_dn = anom_score[(fcnt - self.anom_win[self.current_state, 1]): fcnt]
    label_up = clu_label[0, (fcnt - self.lrn_win[self.current_state, 0]):fcnt]
    label_dn = clu_label[0, (fcnt - self.lrn_win[self.current_state, 1]):fcnt]

    current_clu_score = None
    if(current_label is not None):
        current_clu_score = clu_score[current_label, fcnt]

    # state machine
    if(self.current_state == self.state['normal']):
        '''
         Normal
        '''
        uid = None
        infer = self.state['normal']

        # anom condition - up - windowed score above threshold
        if(np.all(score_up > self.anom_score_thresh)):
            if(self.debug): print('{0}: Norm->Abn, @anom'.format(fcnt))
```

```
            self.next_state = self.state['abnormal']

        # are we in a cluster
        if(current_label is not None):
            # clu condition - up - windowed label the same and score above threshold
            if(np.all(label_up == current_label)
                    and (self.state_counter > self.lrn_win[self.current_state, 0])
                    and (current_clu_score > self.lrn_score_min)
                    and (roll <= self.lrn_cond[self.current_state, 0])):

                if(self.debug): print('{0}: Norm->Abn, @clu'.format(fcnt))
                self.next_state = self.state['abnormal']

    elif(self.current_state == self.state['abnormal']):
        '''
         Abnormal
        '''
        uid = anom_uid[fcnt]
        infer = self.state['abnormal']

        # anom condition - down - windowed score below threshold
        if(np.all(score_dn < self.anom_score_thresh)):
            if(self.debug): print('{0}: Abn->Norm, @anom'.format(fcnt))
            self.next_state = self.state['normal']

        # are we in a cluster
        if(current_label is not None):
            # clu condition - up -current score above threshold and highest label
            if(current_clu_score > self.lrn_score_min
                    and np.any(current_label == label_byscore_asc[-2:])):

                if(self.debug): print('{0}: Abn->Dis, @clu'.format(fcnt))
                self.next_state = self.state['distracted']

    elif(self.current_state == self.state['distracted']):
        '''
         Distracted
```

```
        '''
        uid = anom_uid[fcnt]
        infer = self.state['distracted']

        # anom condition - down - windowed score below threshold
        if(np.all(score_dn < self.anom_score_thresh)
                and (self.state_counter > self.anom_win[self.current_state, 1])):

            if(self.debug): print('{0}: Dis->Norm, @anom'.format(fcnt))
            self.next_state = self.state['normal']

        # are we in a cluster
        if(current_label is not None):
            # clu condition - down - score below threshold
            if(current_clu_score < self.lrn_score_min):
                if(self.debug): print('{0}: Dis->Norm, @clu'.format(fcnt))
                self.next_state = self.state['normal']

            # clu condition - down - label change after win and not in top
            if(np.any(label_dn != current_label)
                    and not np.any(current_label == label_byscore_asc[-3:])
                    and (self.state_counter > self.lrn_win[self.current_state, 1])
                    and (roll <= self.lrn_cond[self.current_state, 1])):

                if(self.debug): print('{0}: Dis->Norm, @clu'.format(fcnt))
                self.next_state = self.state['normal']

            # clu condition - stay - label is the same - will override above conditionals
            if(np.all(label_up == current_label)
                    and (roll <= self.lrn_cond[self.current_state, 0])):

                if(self.debug): print('{0}: Dis->Norm, @clu'.format(fcnt))
                self.next_state = self.state['normal']

    else:
        uid = None
        infer = None
```

```
    # on state change
    self.state_counter = self.state_counter + 1
    if(self.current_state != self.next_state):
        # prevent state change if below min frame time
        if(self.state_counter >= self.state_trans_frame[self.current_state, self.
            next_state]):
            self.state_counter = 0
        else:
            self.next_state = self.current_state

    # create history
    self.hist_infer.update(0, fcnt, infer)

    # change state
    self.current_state = self.next_state

    return uid, infer

def get_pred(self, frame_end):
    if(self.hist_infer.array is not None):
        infer = self.hist_infer.array[0, :frame_end]

        infer[np.isnan(infer)] = 0
        infer[infer == self.state['abnormal']] = 1
        infer[infer == self.state['distracted']] = 1
    else:
        infer = None

    return infer


def plot(self, prefix, features, frame_end):
    """ Plot important metrics ;) """

    # create plots
    y = features.feat['pos_w'].array[0, :frame_end]
```

```
        t = np.linspace(0, y.shape[0], y.shape[0])

        # pull out simple arrays
        # these arrays may not exist
        if(self.hist_infer.array is not None):
            hi = self.hist_infer.array[0, :frame_end]
        else:
            hi = np.zeros(y.shape[0])

        ###
        # FIGURE 0
        ###
        fig = plt.figure(0)
        axs = plt.subplot2grid((1,1), (0,0), rowspan=1)
        axs.plot(hi, linewidth=5)
        axs.set_xlim([0, frame_end])
        axs.set_ylim([-1, 3])
        axs.set_yticks(np.arange(3))
        axs.set_yticklabels(['normal', 'abnormal', 'distracted'])
        axs.set_ylabel('Inference')
        axs.set_xlabel('Frames')
        axs.grid(True)

        fig.tight_layout()
        plt.savefig('{0}/{1}_behav_infer.png'.format(prefix, self.trk), dpi=300)
        plt.close()
```

```
#!/usr/bin/env python3

# default
import os
import sys
import time
import argparse
import datetime
```

```
import warnings
import csv

# from pip
import numpy as np
import pandas as pd
from sklearn.manifold import TSNE
from sklearn import cluster, datasets, mixture
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
from itertools import cycle, islice
# only supports saving to a file
# otherwise we get GTK conflicts with openCV
import matplotlib as mpl
#mpl.use('AGG')
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
import matplotlib.cm as cm

# from git
sys.path.append('../utils/catch22/wrap_Python/catch22')
import catch22_C as c22

# from this
from ObjectFeatures import ObjFeatures, HistArray

class ClusterSeries():

    def __init__(self, params, track, debug=False):
        self.sub_features = self.gen_sub_feature_dict()
        self.window_size = params['sub_features']['window']
        self.trk = track
        self.start_score = 1

        # cluster information
        self.clust_size_max = params['cluster']['size_max']
        self.clust_size_min = params['cluster']['size_min']
```

```
        self.clust_amt = params['cluster']['amount']
        self.cluster_reset()

        # history
        self.hist_labels = HistArray(rows=1, dtype=np.uint16)
        self.hist_size = HistArray(rows=self.clust_amt)
        self.hist_score = HistArray(rows=self.clust_amt)

        # cluster metrics
        self.hist_inertia = HistArray(rows=1)

        # ploting
        self.clust_cmap = plt.get_cmap('tab10')
        self.clust_cmap.set_under('black')
        self.clust_norm = mpl.colors.Normalize(vmin=0, vmax=10)

    def cluster_reset(self):
        # setup cluster
        self.cluster = cluster.MiniBatchKMeans(n_clusters=self.clust_amt,
                                               random_state=0,
                                               reassignment_ratio=0.02)
        self.clust_begin = False
        self.clust_sf_index = []

        # accumulated features
        self.sf = pd.DataFrame(columns=list(self.sub_features.keys()))
        self.labels = None

    def gen_sub_feature_dict(self):
        """
        """
        #TODO think about adding the time series vector itself as a sub feature

        c22_sub_feature_list = [ 'DN_HistogramMode_5',
                                 'DN_HistogramMode_10',
                                 'CO_f1ecac',
                                 'CO_FirstMin_ac',
```

```
                            'CO_HistogramAMI_even_2_5',
                            'CO_trev_1_num',
                            'MD_hrv_classic_pnn40',
                            'SB_BinaryStats_mean_longstretch1',
                            'SB_TransitionMatrix_3ac_sumdiagcov',
                            'PD_PeriodicityWang_th0_01',
                            'CO_Embed2_Dist_tau_d_expfit_meandiff',
                            'IN_AutoMutualInfoStats_40_gaussian_fmmi',
                            'FC_LocalSimple_mean1_tauresrat',
                            'DN_OutlierInclude_p_001_mdrmd',
                            'DN_OutlierInclude_n_001_mdrmd',
                            'SP_Summaries_welch_rect_area_5_1',
                            'SB_BinaryStats_diff_longstretch0',
                            'SB_MotifThree_quantile_hh',
                            'SC_FluctAnal_2_rsrangefit_50_1_logi_prop_r1',
                            'SC_FluctAnal_2_dfa_50_1_2_logi_prop_r1',
                            'SP_Summaries_welch_rect_centroid',
                            'FC_LocalSimple_mean3_stderr']

    # generate dict of function calls
    feat_dict = {}
    for f in c22_sub_feature_list:
        feat_dict['c22_' + f] = getattr(c22, f)

    return feat_dict


def calc_sub_features(self, signal):
    """
        Calculate sub features of a signal
    """
    sf = pd.DataFrame(0, columns=list(self.sub_features.keys()), index=[0])

    # interpolate missing information
    # if all are nans return None
    mask = np.isfinite(signal)
    if(mask.any()):
```

```
            si = np.arange(len(signal))
            signal = np.interp(si, si[mask], signal[mask])

            # shift mean to 0
            signal = signal - np.mean(signal)

            # calculate
            for key, value in self.sub_features.items():
                try:
                    sf[key] = value(signal.tolist())
                except:
                    print(signal)
        else:
            sf = None

        return sf

    '''
        Limit the size of points for each prediction, but keep
        cluster structure:
            - remove outliers (high varrience)
            - points from low scoreing clusters
    '''
    def prune_sub_features(self, labels):
        #TODO
        pass

    def update(self, fcnt, features, score):
        #TODO calculate for all features ...or not
        #TODO lock down cluster for prediciton only at some point
        if(fcnt < self.window_size):
            return

        f = features.feat['pos_w']
        if(f.array is None):
            return
```

```
# get subfeatures from signal window
s = f.array[self.trk, (fcnt-self.window_size):fcnt]
sf = self.calc_sub_features(s)
# if a window is not valid (nans (no track))
# skip adding it to cluster
if(sf is None):
    return

sf = sf.fillna(0)
self.sf = self.sf.append(sf, ignore_index=True)
self.clust_sf_index.append(fcnt)

# once some anomolous behavior is detected
# start to cluster
if(score.array[self.trk, fcnt] > self.start_score and
        len(self.sf.index) > self.clust_size_min):
    self.clust_begin = True

if(self.clust_begin):
    if(self.labels is None):
        # cluster all sub features
        self.labels = self.cluster.fit_predict(self.sf)
    else:
        self.cluster.partial_fit(sf)
        label = self.cluster.predict(sf)
        self.labels = np.append(self.labels, label)

    # find score from when clustering window starts
    sscore = score.array[self.trk, self.clust_sf_index]

    # loop through clusters and update size and score
    for i in range(self.clust_amt):
        label_score = np.mean(sscore[self.labels==i])
        label_count = np.sum(self.labels==i)

        self.hist_score.update(i, fcnt, label_score)
        self.hist_size.update(i, fcnt, label_count)
```

```
        # save off information
        self.hist_labels.update(0, fcnt, self.labels[-1])
        self.hist_inertia.update(0, fcnt, self.cluster.inertia_)


def plot(self, prefix, features, frame_end):
    # check is sf exisits (no object found)
    if(self.sf.size < 2):
        return

    # create plots
    y = features.feat['pos_w'].array[self.trk, :frame_end]
    t = np.linspace(0, y.shape[0], y.shape[0])
    tsne = TSNE(n_components=2, perplexity=15).fit_transform(self.sf)

    # pull out simple arrays
    # these arrays may not exist
    if(self.hist_labels.array is not None):
        cc = self.hist_labels.array[0, :frame_end]
    else:
        cc = np.zeros(y.shape[0])

    if(self.hist_inertia.array is not None):
        ci = self.hist_inertia.array[0, :frame_end]
    else:
        ci = np.zeros(y.shape[0])

    if(self.hist_inertia.array is not None):
        sl = self.hist_score.array[:, :frame_end]
    else:
        sl = np.zeros(shape=(self.clust_amt, y.shape[0]))
    ###
    # FIGURE 0
    ###
    fig = plt.figure(0)
    axs = plt.subplot2grid((5,1), (0,0), rowspan=2)
```

```
# plot signal by class and mark distraction
ccna = cc
ccna[np.isnan(ccna)] = -1
color = self.clust_cmap(self.clust_norm(ccna))

axs.scatter(t, y, c=color, s=10)
axs.set_xlim([0, frame_end])
axs.set_ylabel('Position [Pixels]')
axs.set_xlabel('Frames')
axs.grid(True)

# plot 2D cluster space
axs = plt.subplot2grid((5,1), (2,0), rowspan=3)

ccw = cc[self.clust_sf_index]
ccu = np.unique(ccw)
ccu = ccu[~np.isnan(ccu)]
for c in ccu:
    idx = (c == ccw)
    color = [self.clust_cmap(self.clust_norm(c))]*np.sum(idx)
    axs.scatter(tsne[idx, 0], tsne[idx, 1], c=color,
                label='{:1.0f}'.format(c), s=10)

axs.set_aspect('equal')
axs.set_ylabel('Reduced Y')
axs.set_xlabel('Reduced X')
axs.legend(loc='center left', bbox_to_anchor=(-1, 0.5), title='Cluster Label')
axs.grid(True)

fig.tight_layout()
plt.savefig('{0}/{1}_tsne_clust.png'.format(prefix, self.trk), dpi=300)
plt.close()

###
# FIGURE 1
###
fig = plt.figure(1)
```

```
dfaxs = self.sf.plot(subplots=True, layout=(11,2), figsize=(16,22), xlim=([0,
    frame_end]),
              title='Sub Features')

for dfax in dfaxs:
    dfax[0].legend(loc='upper right')
    dfax[1].legend(loc='upper right')

plt.savefig('{0}/{1}_sub_feat.png'.format(prefix, self.trk), dpi=300)
plt.close()

###
# FIGURE 2
###
fig = plt.figure(2)
plt.bar(t, ci)
plt.xlim([0, frame_end])
axs.set_ylabel('Inertia')
axs.set_xlabel('Frames')
axs.grid(True)

plt.savefig('{0}/{1}_inertia.png'.format(prefix, self.trk), dpi=300)
plt.close()

###
# FIGURE 3
###
fig = plt.figure(3)
for i in range(sl.shape[0]):
    plt.plot(t, sl[i], label='{:1.0f}'.format(i))
plt.xlim([0, frame_end])
plt.legend(loc='upper left', title='Cluster Label')
plt.ylabel('Score')
plt.xlabel('Frames')
axs.grid(True)

plt.savefig('{0}/{1}_cluster_score.png'.format(prefix, self.trk), dpi=300)
```

```
    plt.close()


def debug_run(self, pkl_filepath):
    df = pd.read_pickle(pkl_filepath)
    ittr = len(df.index)

    # only import one track at a time
    features = ObjFeatures(max_track=1,
                           is_live=False,
                           buffer_size=ittr)
    score = HistArray(rows=1,
                          dtype=np.uint8)

    # act on features as if running live
    for index, row in df.iterrows():
        features.update(row['uid'], self.trk, index,
                        row['area'], (row['pos_w'], row['pos_h']))
        score.update(self.trk, index, row['score'])

        # run cluter
        self.update(index, features, score)

    # save off features with updated new score
    prefix_og = './rerun_lrn'
    prefix = prefix_og
    i = 0
    while(os.path.exists(prefix)):
        prefix = prefix_og + '_{0}'.format(i)
        i = i + 1

    os.mkdir(prefix)
    self.plot(prefix, features, ittr)

'''
 Modified from:
    https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html
```

```
'''
def cluster_test(self, X, tsne, signal):
    params = {'quantile': .3,
              'eps': .3,
              'damping': .9,
              'preference': -200,
              'n_neighbors': 10,
              'n_clusters': 3,
              'min_samples': 20,
              'xi': 0.05,
              'min_cluster_size': 0.1}

    # estimate bandwidth for mean shift
    bandwidth = cluster.estimate_bandwidth(X, quantile=params['quantile'])

    # connectivity matrix for structured Ward
    connectivity = kneighbors_graph(
        X, n_neighbors=params['n_neighbors'], include_self=False)
    # make connectivity symmetric
    connectivity = 0.5 * (connectivity + connectivity.T)

    # ============
    # Create cluster objects
    # ============
    ms = cluster.MeanShift(bandwidth=bandwidth, bin_seeding=True)
    two_means = cluster.MiniBatchKMeans(n_clusters=params['n_clusters'])
    ward = cluster.AgglomerativeClustering(
        n_clusters=params['n_clusters'], linkage='ward',
        connectivity=connectivity)
    spectral = cluster.SpectralClustering(
        n_clusters=params['n_clusters'], eigen_solver='arpack',
        affinity="nearest_neighbors")
    dbscan = cluster.DBSCAN(eps=params['eps'])
    optics = cluster.OPTICS(min_samples=params['min_samples'],
                            xi=params['xi'],
                            min_cluster_size=params['min_cluster_size'])
    affinity_propagation = cluster.AffinityPropagation(
```

```
    damping=params['damping'], preference=params['preference'])
average_linkage = cluster.AgglomerativeClustering(
    linkage="average", affinity="cityblock",
    n_clusters=params['n_clusters'], connectivity=connectivity)
birch = cluster.Birch(n_clusters=params['n_clusters'])
gmm = mixture.GaussianMixture(
    n_components=params['n_clusters'], covariance_type='full')

clustering_algorithms = (
    ('MiniBatchKMeans', two_means),
    ('AffinityPropagation', affinity_propagation),
    ('MeanShift', ms),
    ('SpectralClustering', spectral),
    ('Ward', ward),
    ('AgglomerativeClustering', average_linkage),
    ('DBSCAN', dbscan),
    ('OPTICS', optics),
    ('Birch', birch),
    ('GaussianMixture', gmm)
)

for name, algorithm in clustering_algorithms:
    t0 = time.time()

    # catch warnings related to kneighbors_graph
    with warnings.catch_warnings():
        warnings.filterwarnings(
            "ignore",
            message="the number of connected components of the " +
            "connectivity matrix is [0-9]{1,2}" +
            " > 1. Completing it to avoid stopping the tree early.",
            category=UserWarning)
        warnings.filterwarnings(
            "ignore",
            message="Graph is not fully connected, spectral embedding" +
            " may not work as expected.",
            category=UserWarning)
```

```
    algorithm.fit(X)

t1 = time.time()
if hasattr(algorithm, 'labels_'):
    y_pred = algorithm.labels_.astype(np.int)
else:
    y_pred = algorithm.predict(X)

colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00', '#4daf4a',
                                     '#f781bf', '#a65628', '#984ea3',
                                     '#999999', '#e41a1c', '#dede00']),
                              int(max(y_pred) + 1))))
# add black color for outliers (if any)
colors = np.append(colors, ["#000000"])



#Class create plots
y = signal
cc = colors[y_pred]
t = np.linspace(0, y.shape[0], y.shape[0])

fig, axs = plt.subplots(2, 1)
fill = np.ones(y.shape[0] - cc.shape[0])
axs[0].scatter(t, y, c=cc)
axs[0].set_ylabel('Position [Pixels]')
axs[0].set_xlabel('Frames')
axs[0].grid(True)

ccu = np.unique(cc)
for i, color in enumerate(ccu):
    idx = (color == cc)
    axs[1].scatter(tsne[idx, 0], tsne[idx, 1], c=color, label='cluster {0}'.
        format(i))
axs[1].set_ylabel('Reduced Y')
axs[1].set_xlabel('Reduced X')
axs[1].set_aspect('equal')
axs[1].grid(True)
```

```
            axs[1].legend(loc="upper right")

            plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
                     transform=plt.gca().transAxes, size=15,
                     horizontalalignment='right')

            fig.tight_layout()
            plt.savefig('./plots/{0}_spec.png'.format(name), dpi=500)




def main():
    # parse command line arguments
    parser = argparse.ArgumentParser()

    # arguments
    parser.add_argument('-p', '--rerun',
                        help='Load feature data frame (provide path)')

    parser.add_argument('-bp', '--book',
                        help='Load feature data frame and run a bunch\
                                of different algorithms (provide path)')

    args = parser.parse_args()

    lrn = ClusterSeries(None, track=0) #TODO parameters

    if(args.rerun is not None):
        lrn.debug_run(args.rerun)

    elif(args.book is not None):
        # loop through data frame as if we were running live
        # when running live we will convert from features to a data frame
        # of size window_size
        test = pd.read_pickle(args.book)
        test_window = lrn.window_size
```

```
        sub_features = pd.DataFrame(columns=list(lrn.sub_features.keys()))
        # for book options lets pretend uid doesn't matter
        for i in range(test.shape[0] - test_window):
            sub_test = test.loc[(i-test_window):i, "pos_w"]
            sf = lrn.calc_sub_features(sub_test)

            sub_features = sub_features.append(sf, ignore_index=True)

        # fill na and normalize
        sub_features = sub_features.fillna(0)
        sub_features_OG = sub_features
        sub_features = StandardScaler().fit_transform(sub_features)

        # t-SNE
        XE = TSNE(n_components=2, perplexity=15).fit_transform(sub_features)

        # thow the book at it
        os.mkdir('./plots')
        lrn.cluster_test(sub_features, XE, test.loc[test_window:, "pos_w"])


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

```
'''
    Wrapper for object detection
'''
import csv
import cv2
import time
import queue
import numpy as np

from edgetpu.detection.engine import DetectionEngine
```

```
from PIL import Image

from AnomalyScore import ObjFeatures

class ObjectItem():
    def __init__(self, bounding_box, label, score):
        self.bbox = bounding_box
        self.label = label
        self.score = score

        # only track certain things
        # with a certain confidence
        if(((label == "car") or
           (label == "truck") or
           (label == "train")) and
           (score > 0.40)):
            self.track = True
        else:
            self.track = False

        # calculate centroid
        if(self.track):
            sX = self.bbox[0]
            sY = self.bbox[1]
            eX = self.bbox[2]
            eY = self.bbox[3]

            cX = int((sX + eX) / 2.0)
            cY = int((sY + eY) / 2.0)

            self.centroid = (cX, cY)

        else:
            self.centroid = (0, 0)

        # focus is True when this box is directly
        # in front of us
```

```
        self.focus = False

        # set true if being tracked
        self.tracked = False
        self.uid = 0
        self.tid = 0

        # behavior
        self.abnormal = False
        self.distracted = False

    def update(self, uid, cent, tid):
        self.tracked = True
        self.uid = uid
        self.tid = tid
        self.centroid = cent

class ObjectDetect():

    def __init__(self, w, h, min_thresh=0.45, max_det=30, debug=False):

        self.labels_fp = '../models/coco_labels.txt'
        self.network_fp = '../models/mobilenet_ssd_v2_coco_quant_postprocess_edgetpu.tflite'
        self.ids_to_keep_fp = '../models/coco_keep.csv'

        self.debug = debug
        self.bbox_max_w = 0.9 * w
        self.bbox_max_h = 0.9 * h
        self.min_thresh = min_thresh
        self.max_det = max_det

        # read in labels for detector
        with open(self.labels_fp, 'r', encoding="utf-8") as f:
            lines = f.readlines()
        self.labels = {}
        for line in lines:
            pair = line.strip().split(maxsplit=1)
```

```
            self.labels[int(pair[0])] = pair[1].strip()

        # read in labels to keep
        self.ids_to_keep = []
        with open(self.ids_to_keep_fp, 'r') as csvFile:
            for row in csv.reader(csvFile):
                self.ids_to_keep.append(int(row[0]))

        # initialize engine
        self.ssd_engine = DetectionEngine(self.network_fp)

    '''
        img is as an openCV image
        returned filtered object box descriptions
    '''
    def run(self, rgb):
        # convert to PIL format
        im_pil = Image.fromarray(rgb)

        # run pipeline
        obj_raw = self.detect_objects(im_pil)
        obj_det = self.object_processing(obj_raw)

        return obj_det

    def run_tr(self, in_q, out_q):
        while(True):
            # pull from queue
            rgb = in_q.get()

            if(rgb is not None):
                # run detect
                det_objs = self.run(rgb)
                out_q.put(det_objs)

    def detect_objects(self, image):
        # run inference
```

```
    obj_det = self.ssd_engine.DetectWithImage(image,
                                              threshold=self.min_thresh,
                                              keep_aspect_ratio=False,
                                              relative_coord=False,
                                              top_k=self.max_det)
    return obj_det

'''
 Filter objects we don't care about both by
    label and size
'''
def object_processing(self, obj_det):
    obj_itm = []

    # only want to loop once
    if obj_det:
        for obj in obj_det:
            # filter by label
            if not (obj.label_id in self.ids_to_keep):
                continue

            # filter by bbox size
            box = obj.bounding_box.astype(int).flatten().tolist()
            box_w = box[2] - box[0]
            box_h = box[3] - box[1]

            if((box_w > self.bbox_max_w)
                   or (box_h > self.bbox_max_h)):
                continue

            # add object to list
            item = ObjectItem(box, self.labels[obj.label_id],
                              obj.score)
            obj_itm.append(item)

    return obj_itm
```

```
'''
    Keeping track of features thoughout run
    and a helper class to handle array expansion
'''
import numpy as np


'''
    automatically handle buffer expansion
'''
class HistArray():
    def __init__(self, rows, dtype=np.float32, keep=True, buffer_size=(3*60*30)):
        self.array = None
        self.rows = rows
        self.dtype = dtype
        self.keep = keep
        self.clip = 0

        # default 3 minutes by 60 seconds by 30 fps
        self.buffer_size = buffer_size

    def update(self, row, idx, val):
        # manage size
        if(self.array is None):
            self.array = np.zeros(shape=(self.rows, self.buffer_size),
                                  dtype=self.dtype) * np.nan
        else:
            # do we need to expand in time
            if(self.array.shape[1] < (idx+1)):
                if(self.keep):
                    bf = np.zeros(shape=(self.rows, self.buffer_size),
                                          dtype=self.dtype) * np.nan
                    self.array = np.append(self.array, bf, axis=1)

                else:
                    self.clip = idx
```

```
                # keep the last half and add new buffer
                bf = np.zeros(shape=(self.rows, (self.buffer_size // 2)),
                                    dtype=self.dtype) * np.nan
                self.array = np.append(self.array[:,:(self.array.shape[1] // 2)], bf,
                    axis=1)

        # add value
        if(self.keep):
            self.array[row, idx] = val
        else:
            self.array[row, idx - self.clip] = val

'''
    organize all features by frame
'''
class ObjFeatures():
    def __init__(self, max_track, is_live=False, buffer_size=(3*60*30)):
        # settings
        self.max_track = max_track
        self.is_live = is_live

        # smoothing
        self.smooth_win = 8
        win = np.hanning(self.smooth_win)
        self.norm_win = win/win.sum()

        # values
        self.feature_list = ["area", "pos_w", "pos_h"]

        self.uid_key = HistArray(rows=max_track, keep=(not is_live), buffer_size=buffer_size
            )

        self.feat = {}
        self.sfeat = {}
        self.dfeat = {}
        for fkey in self.feature_list:
```

```
        self.feat[fkey] = HistArray(rows=max_track, keep=(not is_live), buffer_size=
            buffer_size)
        self.sfeat[fkey] = HistArray(rows=max_track, keep=(not is_live), buffer_size=
            buffer_size)
        self.dfeat[fkey] = HistArray(rows=max_track, keep=(not is_live), buffer_size=
            buffer_size)


def update(self, uid, tid, fcnt, area, dis):
    # add values
    self.uid_key.update(tid, fcnt, uid)
    self.feat['area'].update(tid, fcnt, area)
    self.feat['pos_w'].update(tid, fcnt, dis[0])
    self.feat['pos_h'].update(tid, fcnt, dis[1])

    for key, ft in self.feat.items():
        for i in range(ft.array.shape[0]):

            if(fcnt > self.smooth_win):
                # smooth values
                weights = self.norm_win * ft.array[i, (fcnt - self.smooth_win):fcnt]
                smooth = np.sum(weights)
                # note this is delayed by self.smooth_win // 2
                self.sfeat[key].update(i, fcnt, smooth)

                # take derivative
                if(fcnt == self.smooth_win):
                    self.dfeat[key].update(i, fcnt, smooth)
                else:
                    last = self.sfeat[key].array[i,fcnt-1]
                    self.dfeat[key].update(i, fcnt, smooth - last)


def gen_features(self, objs, frame_count):
   for obj in objs:
       if(obj.tracked):
           # create features
```

```
            area = (obj.bbox[2] - obj.bbox[0]) * \
                   (obj.bbox[3] - obj.bbox[1])

            self.update(obj.uid, obj.tid,
                        frame_count,
                        area, obj.centroid)
```

```
'''
    Wrapper for object tracking

    inspired from:
        https://www.pyimagesearch.com/2018/10/29/multi-object-tracking-with-dlib/
        https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/
'''
import cv2
import time
import dlib
import numpy as np
import multiprocessing
from scipy.spatial import distance as dist
from collections import OrderedDict

from ObjectDetect import ObjectItem

class ObjectTrack():

    def __init__(self, frames_till_forget=10, max_distance=50,
                 tracker_limit=4, min_track_corr=20, debug=False):
        # params
        self.debug = debug
        self.tracker_limit = tracker_limit
        self.frames_till_forget = frames_till_forget
        self.max_distance = max_distance
        self.min_corr = min_track_corr
```

```
        # init
        #TODO muliprocessing
        #self.in_track_queues = []
        #self.out_track_queues = []

        # id and objects we are keeping track of
        self.next_id = 0
        self.trackers = OrderedDict()
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()

    '''
     This section is where we update the tracks with the image
      and update the object ids
    '''
    def update_objects(self, objs, rgb):
        trk = []

        # filter for relevant objects
        for obj in objs:
            if(obj.track):
                trk.append(obj)

        trk_ass = self.update(trk, rgb)

        return list(self.objects.values()), trk_ass

    def update_frame(self, rgb, alt=False):
        objs = []
        derid = []

        # loop through trackers and objects then update position
        for i, (track, obj) in self.trackers.items():
            corr = track.update(rgb)

            # deregister if we are not well corrilated
            if(corr > self.min_corr):
```

```
            pos = track.get_position()
            obj.bbox[0] = int(pos.left())
            obj.bbox[1] = int(pos.top())
            obj.bbox[2] = int(pos.right())
            obj.bbox[3] = int(pos.bottom())

            cX = int((obj.bbox[0] + obj.bbox[2]) / 2.0)
            cY = int((obj.bbox[1] + obj.bbox[3]) / 2.0)

            self.objects[obj.uid].bbox = obj.bbox
            self.objects[obj.uid].centroid = (cX, cY)

        else:
            derid.append(obj.uid)

    # deregister (can't remove during rotation)
    for oid in derid:
        self.deregister(oid)

    return list(self.objects.values())

def update_all(self, objs, rgb):

    # run object association
    (u_objs, trk_ass) = self.update_objects(objs, rgb)

    # if the ids of the associations equal the tracked
    # object ids then there is no need to run the tracker update
    trk_ids = list(self.objects.keys())
    id_check = all(anid in trk_ids for anid in trk_ass)

    if(id_check):
        u_objs = self.update_frame(rgb, True)

    return u_objs
```

```
    '''
     This section of functions handles the association and
        creation of ids
    '''
    def register(self, obj, cent, rgb):

        # start up a tracker
        amt_trks = len(self.trackers)
        if(amt_trks < self.tracker_limit):
            # find available trk ids
            all_ids = np.arange(self.tracker_limit).tolist() #TODO probably a better way
            trk_ids = [oo.tid for oo in list(self.objects.values())]
            avail_ids = list(set(all_ids) - set(trk_ids))

            # update object with id information
            obj.update(self.next_id, cent, avail_ids[0])

            # start up tracker
            t = dlib.correlation_tracker()
            rect = dlib.rectangle(obj.bbox[0], obj.bbox[1],
                                  obj.bbox[2], obj.bbox[3])
            t.start_track(rgb, rect)

            # save off
            self.objects[self.next_id] = obj
            self.disappeared[self.next_id] = 0
            self.trackers[self.next_id] = (t, obj)

            # get next id
            self.next_id += 1

    def deregister(self, oid):
        del self.objects[oid]
        del self.disappeared[oid]
        del self.trackers[oid]

    def update(self, objs, rgb):
```

```
trk_ass = []

# if empty no association requiered
if(len(objs) == 0):

    # mark objects as missing a detection
    oids_to_remove = []
    for oid in self.disappeared.keys():
        self.disappeared[oid] += 1

        # remove if missing for too long
        if(self.disappeared[oid] > self.frames_till_forget):
            oids_to_remove.append(oid)

    for oid in oids_to_remove:
        self.deregister(oid)

    return trk_ass

# calculate centroids
in_centroids = np.array([x.centroid for x in objs])

# register centroids if nothing is currently being tracked
if(len(self.objects) == 0):
    for i in range(len(in_centroids)):
        self.register(objs[i], in_centroids[i], rgb)

else:
    object_ids = list(self.objects.keys())
    tobjs = list(self.objects.values())
    object_cen = np.array([x.centroid for x in tobjs])

    # calculate distances between tracked objects and new centroids
    dc = dist.cdist(object_cen, in_centroids)

    # order by the smallest distances
    rows = dc.min(axis=1).argsort()
```

```
cols = dc.argmin(axis=1)[rows]

used_rows = set()
used_cols = set()

# loop over the distances and look for tracked objects
# being close to new centroids
for (row, col) in zip(rows, cols):
    if((row in used_rows) or (col in used_cols)):
        continue

    # if the distance is under a certain amount we associate
    # the centroid with the object
    if(dc[row,col] < self.max_distance):
        object_id = object_ids[row]

        # update
        self.objects[object_id].centroid = in_centroids[col]
        self.objects[object_id].bbox = objs[col].bbox
        self.disappeared[object_id] = 0

        # count number of associations
        trk_ass.append(object_id)

        # mark row and col as used
        used_rows.add(row)
        used_cols.add(col)

# which ones are we missing
unused_rows = set(range(dc.shape[0])).difference(used_rows)
unused_cols = set(range(dc.shape[1])).difference(used_cols)

# check for objects who disappeared
if(dc.shape[0] >= dc.shape[1]):
    for row in unused_rows:
        object_id = object_ids[row]
        self.disappeared[object_id] += 1
```

```
                if(self.disappeared[object_id] > self.frames_till_forget):
                    self.deregister(object_id)

            # call it a new object
            else:
                for col in unused_cols:
                    self.register(objs[col], in_centroids[col], rgb)

        return trk_ass


"""
# for multi processing later #TODO
def start_tracker(obj, rgb, input_q, output_q):
    # get tracker from dlib
    t = dlib.correlation_tracker()
    rect = dlib.rectangle(obj.bbox[0], obj.bbox[1], obj.bbox[2], obj.bbox[3])

    t.start_track(rgb, rect)

    # keep looking for frame updates
    while(True):
        rgb = input_q.get()

        # get and update position
        if(rgb is not None):
            t.update(rgb)

            pos = t.get_position()
            obj.bbox[0] = int(pos.left())
            obj.bbox[1] = int(pos.top())
            obj.bbox[2] = int(pos.right())
            obj.bbox[3] = int(pos.bottom())

            output_q.put(obj)
"""
```

```
'''
 implementation modified from:
 Jean-Paul van Brakel via:
 https://stackoverflow.com/questions/22583391/peak-signal-detection-in-realtime-timeseries-
     data/22640362#22640362
'''
import numpy as np

# only supports saving to a file
# otherwise we get GTK conflicts with openCV
import matplotlib
matplotlib.use('AGG')
import matplotlib.pyplot as plt

class Zscore():
    def __init__(self, lag, threshold, influence, memory_size=1):

        self.lag = lag
        self.threshold = threshold
        self.influence = influence
        self.debug = False

        self.init = [False] * memory_size
        self.filtered = np.zeros(shape=(memory_size, self.lag))
        self.avg_filter = np.zeros(shape=(memory_size, 1))
        self.std_filter = np.zeros(shape=(memory_size, 1))

    # give y_lag as size lag
    def run(self, y_lag, mbank):

        # init algorithm
        if(self.init[mbank]):
            self.filtered[mbank,:] = y_lag
            self.avg_filter[mbank] = np.mean(self.filtered[mbank,:])
```

```
        self.std_filter[mbank] = np.std(self.filtered[mbank,:])
        self.init[mbank] = True

    # setup
    found = None
    yy = y_lag[-1]

    # did we excede std deviation threshold
    if(np.abs(yy - self.avg_filter[mbank]) >
            (self.threshold * self.std_filter[mbank])):

        # which way are we over
        if(yy > self.avg_filter[mbank]):
            found = 1
        else:
            found = -1

        # shift
        fyy = (self.influence * yy) + \
                ((1-self.influence) * self.filtered[mbank, -1])
        self.filtered[mbank, :] = np.append(
                    self.filtered[mbank, 1:], fyy)

    else:
        self.filtered[mbank, :] = np.append(
                    self.filtered[mbank, 1:], yy)

    # update
    self.avg_filter[mbank] = np.mean(self.filtered[mbank,:])
    self.std_filter[mbank] = np.std(self.filtered[mbank,:])

    if(self.debug):
        print('in: {0}, avg/std: {1}/{2}, found: {3}'.format(yy, self.avg_filter[mbank],
              self.std_filter[mbank], found))

    return found
```

```
    def plot(self, title, y, mbank):
        fig, axs = plt.subplots(2, 1)

        axs[0].plot(y, color="b", linewidth=1.0)
        axs[0].axhline(y=self.avg_filter[mbank], color="c", linewidth=2.0)
        axs[0].axhline(y=self.avg_filter[mbank] + (self.threshold * self.std_filter[mbank]),
                   color="g", linewidth=1.0)
        axs[0].axhline(y=self.avg_filter[mbank] - (self.threshold * self.std_filter[mbank]),
                   color="g", linewidth=1.0)
        axs[0].set_xlabel('frames')
        axs[0].set_title(title)

        fig.tight_layout()
        plt.show()
```

```
AnomalyScore:
  zscore:
    norm:
        lag: 15
        threshold: 1.8
        influence: 0.8
    diff:
        lag: 60
        threshold: 2.2
        influence: 0.6
  fft:
    window: 90
  score:
    pos_w: 4
    pos_w_fft: 2
    pos_h: 1
    area: 1
  peak:
    min_frame: 15
ClusterSeries:
```

```
  sub_features:
    window: 30
  cluster:
    size_max: 2e3
    size_min: 90
    amount: 6
BehaviorEst:
  anomaly:
    window:
      normal_up: 10
      abnormal_up: 10
      abnormal_down: 10
      distracted_down: 40
    score_threshold: 3
  learn:
    window:
      normal_up: 5
      abnormal_up: 5
      abnormal_down: 5
      distracted_stay: 10
      distracted_down: 20
    probability:
      normal_up: 0.1
      distracted_stay: 1
      distracted_down: 0.1
    score_threshold_ratio: 0.3
```

## C.2 Python Real Time

```
#!/usr/bin/env python3

"""
    The real time (threaded) version of the DDD pipeline
```

```
    #import pdb; pdb.set_trace() #DEBUG
"""

# std
import os
import sys
import time
import argparse
import datetime
import csv
import queue
import threading

# available in pip
import numpy as np
import cv2 as cv
from termcolor import cprint
from pyfiglet import figlet_format

# own code
from ddd import DDD
from DisplayHelper import DisplayHelper

def main():
    # parse command line arguments
    parser = argparse.ArgumentParser()

    # arguments
    parser.add_argument('-r', '--rec_only', action='store_true',
                        help='Store only raw video with no annotations')

    parser.add_argument('-d', '--debug', action='store_true',
                        help='Print extra debug information')

    args = parser.parse_args()

    # welcome text
```

```
cprint(figlet_format('DDD !!!', font='starwars'),
       'yellow', 'on_grey', attrs=['bold'])

# setup objects
dh = DisplayHelper(rec_only=args.rec_only) #t0
drt = DDD()

# start text
cprint(figlet_format('Start', font='cybermedium'),
       'green', 'on_grey', attrs=['bold'])

if(not args.rec_only):
    # setup thread 1
    t1_in_q = queue.Queue(maxsize=1)
    t1_out_q = queue.Queue(maxsize=1)

    # launch t1
    t1 = threading.Thread(target=drt.detect_objects.run_tr,
                          args=(t1_in_q, t1_out_q), daemon=True)
    t1.start()
else:
    # rec only text
    cprint(figlet_format('REC Only...', font='cybermedium'),
           'blue', 'on_grey', attrs=['bold'])

# video loop
while(True):
    # get, display and run detector
    image, rgb = dh.read()

    if(not args.rec_only):
        # pass to queue
        try:
            t1_in_q.put_nowait(rgb)
        except queue.Full:
            # drop current image if full
            pass
```

```
            # pull from queue
            try:
                detections = t1_out_q.get_nowait()
                tracks = drt.tracker.update_all(detections, rgb)
            except queue.Empty:
                detections = None
                tracks = None

            # annotate image
            drt.annotate_display(image, detections, tracks, True)

        # display
        dh.display(image)

        # check for exit
        if dh.close:
            break


    # end text
    cprint(figlet_format('FIN', font='cybermedium'),
           'red', 'on_grey', attrs=['bold'])


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

```
"""
    This class will host the camera, window and video recording information
"""
# std
import os
import sys
import time
```

```
import argparse
import datetime
import csv
import threading

# available in pip
import numpy as np
import cv2 as cv



class DisplayHelper():
    def __init__(self, rec_only=False):
        # settings
        self.window_name = 'ddd'
        self.width = 800
        self.height = 600
        self.vid_fps = 30.0
        self.scn_debug = True
        self.time_last = time.time()
        self.fps = 0
        self.rec_only = rec_only

        if(self.rec_only):
            self.vid_codac = cv.VideoWriter_fourcc(*'XVID')
        else:
            self.vid_codac = cv.VideoWriter_fourcc(*'MJPG')

        # overlay font
        self.font = cv.FONT_HERSHEY_SIMPLEX
        self.font_scale = 0.5
        self.font_thick = 1
        self.font_line = cv.LINE_AA
        self.text_color = (255, 255, 153) # light yellow
        self.back_color = (0, 0, 0) # black

        # time stamp
        time_str = '{:%Y-%b-%d_%H_%M_%S_%f}'.format(datetime.datetime.now())
```

```
    # setup camera
    cap = cv.VideoCapture(0)

    cap.set(cv.CAP_PROP_FRAME_WIDTH, self.width)
    cap.set(cv.CAP_PROP_FRAME_HEIGHT, self.height)
    cap.set(cv.CAP_PROP_FPS, self.vid_fps)
    cap.set(cv.CAP_PROP_AUTOFOCUS, 0) # off
    cap.set(cv.CAP_PROP_FOCUS, 0) # min (255 is max)
    self.__cam = cap
    self.__raw = None
    (self.tr_ret, self.tr_image) = self.__cam.read()

    self.thread_running = True
    threading.Thread(target=self.update, args=()).start()

    # setup display
    cv.namedWindow(self.window_name, cv.WINDOW_AUTOSIZE)
    cv.moveWindow(self.window_name, 0, 0)
    cv.setWindowProperty(self.window_name, cv.WND_PROP_FULLSCREEN,
                         cv.WINDOW_FULLSCREEN)

    # setup recording streams
    if(not self.rec_only):
        self.outa = cv.VideoWriter('{0}_anno_output.avi'.format(time_str),
                        self.vid_codac, self.vid_fps,
                        (self.width, self.height))

    self.outr = cv.VideoWriter('{0}_raw_output.avi'.format(time_str),
                    self.vid_codac, self.vid_fps,
                    (self.width, self.height))

@property
def close(self):
    #is_closed = (cv.getWindowProperty(self.window_name, 0) < 0)
    is_closed = (cv.waitKey(1) & 0xFF == ord('q'))
```

```
        if(is_closed):
            # stop thread
            self.thread_running = False

            # release and destroy
            self.outr.release()
            if(not self.rec_only): self.outa.release()
            self.__cam.release()
            cv.destroyAllWindows()


        return is_closed

    def update(self):
        while(True):
            # check for stopped thread
            if(not self.thread_running):
                return

            # grab next frame
            (self.tr_ret, self.tr_image) = self.__cam.read()


    def read(self):
        if(self.tr_ret):
            # camera is mounted upside-down
            img = cv.flip(self.tr_image, -1)
            rgb = cv.cvtColor(img, cv.COLOR_BGR2RGB)

            # unedited copy for playback
            self.__raw = img.copy()

            # calculate FPS
            now = time.time()
            self.fps = 1 / (now - self.time_last)
            self.time_last = now
        else:
```

```
            img = None

        return img, rgb

    def display(self, image):
        # add debug information
        if(self.scn_debug):
            fps_info = 'FPS: {:.2f}'.format(self.fps)

            # outer box
            text_start = (15, 15)
            text_size, base = cv.getTextSize(fps_info, self.font,
                                             self.font_scale,
                                             self.font_thick)
            text_end = (text_start[0] + text_size[0],
                        text_start[1] - text_size[1])
            cv.rectangle(image,
                         text_start,
                         text_end,
                         self.back_color, cv.FILLED)
            # FPS
            cv.putText(image, fps_info, text_start,
                         self.font, self.font_scale,
                         self.text_color, self.font_thick,
                         self.font_line)

        # display image
        cv.imshow(self.window_name, image)

        # record video
        self.outr.write(self.__raw)
        if(not self.rec_only): self.outa.write(image)
```

## C.3 Python Utilities

```
#!/usr/bin/env python

import glob
import os
import sys
import threading
import random
import time
import shutil
import multiprocessing

try:
    sys.path.append(glob.glob('../carla/dist/carla-*%d.%d-%s.egg' % (
        sys.version_info.major,
        sys.version_info.minor,
        'win-amd64' if os.name == 'nt' else 'linux-x86_64'))[0])
except IndexError:
    pass

import carla
import skvideo.io

try:
    import pygame
except ImportError:
    raise RuntimeError('cannot import pygame, make sure pygame package is installed')

try:
    import numpy as np
except ImportError:
    raise RuntimeError('cannot import numpy, make sure numpy package is installed')

try:
    import queue
except ImportError:
    import Queue as queue
```

```
class DriveAndFilm():

    def __init__(self):
        pygame.init()

        # keys are known working vehicles and list of known good colors
        self.vehicles = {'vehicle.lincoln.mkz2017': [2, 5],
                         'vehicle.nissan.patrol': [0, 2],
                         #'vehicle.audi.a2': [0],
                         'vehicle.tesla.model3': [0]}

        # known good waypoint coordinates pairs
        # 0 - lead car, 1 - tail car : (x, y)
        self.waypoints = {'Town03_loc-0': [(186, -2), (193, -2)],
                          'Town01_loc-0': [(243, 327), (250, 327)],
                          'Town01_loc-1': [(174, 60), (167, 60)]}

        # maneuver variation bounds
        self.drift_bounds = {'duration': np.linspace(1.0, 2.5, 5),
                             'strength': np.linspace(0.07, -0.07, 4)}
        self.swerve_bounds = {'duration': np.linspace(0.4, 1, 5),
                              'strength': np.linspace(0.50, -0.50, 4)}
        print(self.drift_bounds)

    def run(self, seed=42, maneuver_type='drift'):
        # set random seed
        random.seed(a=seed, version=2)
        postfix = maneuver_type + '_' + str(seed)

        # create a new folder for images
        og_image_folder = './saved_images'
        image_folder = og_image_folder
        i = 0
        while(os.path.exists(image_folder)):
            image_folder = og_image_folder + '_{0}'.format(i)
```

```
    i = i + 1

# connect
client = carla.Client('localhost', 2000)
client.set_timeout(5.0)

# connect to world
position_list = [x for x in self.waypoints.keys()]
                #if(x.startswith(wmap.name))]
rkey_pos = random.choice(position_list)
postfix += '_' + rkey_pos

if(rkey_pos.startswith('Town01')):
    world = client.load_world('Town01')

elif(rkey_pos.startswith('Town03')):
    world = client.load_world('Town03')

wmap = world.get_map()
print('using map: ' + wmap.name)

# setup syncronous mode
print('enabling synchronous mode.')
settings = world.get_settings()
settings.synchronous_mode = True
world.apply_settings(settings)
world.tick()

actor_list = []
try:
    blueprint_library = world.get_blueprint_library()

    # select vehicle type
    rkey = random.choice(list(self.vehicles.keys()))
    rcol = random.choice(self.vehicles[rkey])
    print('using vehicle: ' + rkey)
```

```
# setup vehicle type
bp = blueprint_library.find(rkey)
color = bp.get_attribute('color').recommended_values[rcol]
bp.set_attribute('color', color)

# select starting positions
rpos = self.waypoints[rkey_pos]
lead_waypoint = wmap.get_waypoint(
                  carla.Location(x=rpos[0][0], y=rpos[0][1]))
tail_waypoint = wmap.get_waypoint(
                  carla.Location(x=rpos[1][0], y=rpos[1][1]))

# spawn the lead car
lead_vehicle = world.spawn_actor(bp, lead_waypoint.transform)

lead_vehicle.set_simulate_physics(True)
lead_vehicle.set_autopilot(True)
actor_list.append(lead_vehicle)
print('created: {} at {}'.format(lead_vehicle.type_id,
                            lead_waypoint.transform.location))

# spawn the camera car
tail_vehicle = world.spawn_actor(bp, tail_waypoint.transform)

tail_vehicle.set_simulate_physics(True)
actor_list.append(tail_vehicle)
print('created: {} at {}'.format(tail_vehicle.type_id,
                            tail_waypoint.transform.location))

# setup camera
camera_bp = blueprint_library.find('sensor.camera.rgb')
#camera_bp.set_attribute('image_size_x', '800')
#camera_bp.set_attribute('image_size_y', '600')

## set camera capture rate to 30 FPS
#camera_bp.set_attribute('sensor_tick', str(1/30))
```

```
# attach camera
camera_transform = carla.Transform(carla.Location(x=1.5, z=1.5))
camera = world.spawn_actor(camera_bp,
                  camera_transform, attach_to=tail_vehicle)
actor_list.append(camera)
print('created: {} at {}'.format(camera.type_id,
                  camera_transform.location))

# setup camera capture
camera.listen(lambda image:\
    image.save_to_disk(image_folder +\
    '/%06d.png' % image.frame_number))

"""
# set the spectator to spawned location
spectator = world.get_spectator()
spectator.set_transform(tail_waypoint.transform)
"""

"""
#TODO auto pilot settings
# https://carla.readthedocs.io/en/latest/measurements/
measurements, sensor_data = carla_client.read_data()
control = measurements.player_measurements.autopilot_control
# modify here control if wanted.
carla_client.send_control(control)
tail_vehicle.set_autopilot(True)
"""

# Make sync queue for sensor data
camera_bp = blueprint_library.find('sensor.camera.rgb')
camera_transform = carla.Transform(carla.Location(x=-5.5, z=2.8))
camera = world.spawn_actor(camera_bp,
                     camera_transform, attach_to=tail_vehicle)
actor_list.append(camera)
print('created: {} at {}'.format(camera.type_id,
                         camera_transform.location))
```

```
image_queue = queue.Queue()
camera.listen(image_queue.put)

# setup display
frame = None
display = pygame.display.set_mode(
    (800, 600),
    pygame.HWSURFACE | pygame.DOUBLEBUF)
font = self.get_font()

clock = pygame.time.Clock()
start_time = pygame.time.get_ticks()

# setup maneuver
if(maneuver_type == 'drift'):
    bounds = self.drift_bounds
else:
    bounds = self.swerve_bounds

maneuver_start = 7
maneuver_time = random.choice(bounds['duration'])
film_end = 16
et = [x*1000 for x in
        [maneuver_start,
            (maneuver_start + maneuver_time),
            film_end]
     ]
lead_steer = random.choice(bounds['strength'])
print(et)

# loop
while True:
    if self.should_quit():
        return

    # get and advance time
```

```
clock.tick()
world.tick()
ts = world.wait_for_tick()
eltime = (pygame.time.get_ticks() - start_time)

# alter behavior
tail_vehicle.apply_control(
        carla.VehicleControl(throttle=0.6, steer=0.00))

if(maneuver_type == 'drift'):
    # drift
    if(et[0] < eltime < et[1]):
        # execute maneuver
        lead_vehicle.set_autopilot(False)
        lead_vehicle.apply_control(
                carla.VehicleControl(throttle=0.35, steer=lead_steer))

        tail_vehicle.apply_control(
                carla.VehicleControl(throttle=0.0, steer=0.00))

    elif(et[1] < eltime < et[2]):
        # correct
        lead_vehicle.set_autopilot(True)

    elif(et[2] < eltime):
        # end run
        break

else:
    # swerve
    if(et[0] < eltime < et[1]):
        # execute maneuver
        lead_vehicle.set_autopilot(False)
        lead_vehicle.apply_control(
                carla.VehicleControl(throttle=0.5, steer=lead_steer))

        tail_vehicle.apply_control(
```

```
                carla.VehicleControl(throttle=0.0, steer=0.00))

    elif(et[1] < eltime < et[2]):
        # correct
        lead_vehicle.set_autopilot(True)

    elif(et[2] < eltime):
        # end run
        break

# frame management
if frame is not None:
    if ts.frame_count != frame + 1:
        logging.warning('frame skip!')

frame = ts.frame_count

while True:
    image = image_queue.get()
    if image.frame_number == ts.frame_count:
        break
    logging.warning(
        'wrong image time-stampstamp: frame=%d, image.frame=%d',
        ts.frame_count,
        image.frame_number)

self.draw_image(display, image)

fps_surface = font.render('% 5d FPS' % clock.get_fps(),
                          True, (255, 255, 255))
time_surface = font.render(' % 3.2f s' % (eltime / 1000),
                          True, (255, 255, 255))
display.blit(fps_surface, (8, 10))
display.blit(time_surface, (8, 23))

pygame.display.flip()
```

```
    finally:
        print('\ndisabling synchronous mode.')
        settings = world.get_settings()
        settings.synchronous_mode = False
        world.apply_settings(settings)

        # clean up
        print('destroying assests')
        for actor in actor_list:
            actor.destroy()
        print('done.')

    return postfix, image_folder

def exit(self):
    pygame.quit()

def draw_image(self, surface, image):
    array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
    array = np.reshape(array, (image.height, image.width, 4))
    array = array[:, :, :3]
    array = array[:, :, ::-1]
    image_surface = pygame.surfarray.make_surface(array.swapaxes(0, 1))
    surface.blit(image_surface, (0, 0))

def get_font(self):
    fonts = [x for x in pygame.font.get_fonts()]
    default_font = 'ubuntumono'
    font = default_font if default_font in fonts else fonts[0]
    font = pygame.font.match_font(font)
    return pygame.font.Font(font, 14)

def should_quit(self):
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            return True
        elif event.type == pygame.KEYUP:
```

```
                if event.key == pygame.K_ESCAPE:
                    return True
        return False

def img_to_video(in_q):
    while True:
        try:
            # attempt to get the next item
            item = in_q.get_nowait()

        except queue.Empty:
            break

        else:
            # process item
            postfix, folder_path = item

            # find an unused filename
            print('converting images to video')
            og_filename = 'carla_' + postfix
            filename = og_filename
            i = 0
            while(os.path.exists(filename + '.mp4')):
                filename = og_filename + '_{0}'.format(i)
                i = i + 1
            filename += '.mp4'

            # create video writer
            # fps = 30 doesn't seem to matter
            writer = skvideo.io.FFmpegWriter(filename)
            # outputdict={'-r': str(fps)})

            # loop though images
            for (root, _, filenames) in os.walk(folder_path):
                for name in filenames:
                    fname = os.path.join(root, name)
```

```
                if(fname.endswith('.png')):
                    img = skvideo.io.vread(fname)
                    writer.writeFrame(img)
            writer.close()
            shutil.rmtree(folder_path)
            print('finished: {}'.format(postfix))
            time.sleep(.1)

    return True

def main():
    file_q = multiprocessing.Queue()

    # start sim actor
    daf = DriveAndFilm()
    #maneu = 'swerve'
    maneu = 'drift'
    for i in range(127, 138):
        postfix, image_folder = daf.run(seed=i,
                                        maneuver_type=maneu)
        file_q.put((postfix, image_folder))
        time.sleep(10)
    daf.exit()

    # creating processes
    cpu_count = multiprocessing.cpu_count()
    processes = []

    print('multiprocess count: {}'.format(cpu_count // 2))
    for _ in range(cpu_count // 2):
        p = multiprocessing.Process(target=img_to_video, args=(file_q,))
        processes.append(p)
        p.start()

    # completing process
    for p in processes:
        p.join()
```

```
    print('processes completed')


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

```
"""
"""
# default
import os
import sys

# pip
import yaml
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

# own
sys.path.append('../src')
from ddd import auto_batch_test

class D3E(BaseEstimator, ClassifierMixin):
    """ class to cast ddd as a sci-kit learn estimator
    """

    def __init__(self, prefix_in):
        # run dirs
        self.run_folder_path = '../data/video'
        self.truth_file = '../data/truth/labels.json'
        self.prefix_in = prefix_in

        # load default params
```

```
        param_fp = '../src/parameters.yaml'
        with open(param_fp) as f:
            self.params = yaml.load(f, Loader=yaml.FullLoader)

    def set_params(self, **parameters):
        for parameter, value in parameters.items():
            setattr(self, parameter, value)

        # override
        self.params['AnomalyScore']['zscore']['norm']['lag'] = self.zscore_norm_lag
        self.params['AnomalyScore']['zscore']['norm']['threshold'] = self.
            zscore_norm_threshold

        return self

    def fit(self, X, y=None):
        return self

    def predict(self, X, y=None):
        return([])

    def score(self, X, y=None):
        # create a new directory to host all re-runs
        og_prefix = '{}/sub_param_opt'.format(self.prefix_in)
        prefix = og_prefix
        i = 0
        while(os.path.exists(prefix)):
            prefix = og_prefix + '_{0}'.format(i)
            i = i + 1
        os.mkdir(prefix)
        self.prefix = prefix

        # run everything
        report_info = auto_batch_test(self.params, self.prefix,
                                      self.run_folder_path,
                                      self.truth_file)
        # calculate metrics
```

```
        total_mcc = np.mean(np.nan_to_num(np.asarray(
                            [v['MCC'] for k,v in report_info.items()]
                        )))

        # print status
        print('Run: {}\nFolder: {}\n Performance: {}\nParameters: {}'.format(i,
                        self.prefix, total_mcc, self.params))

        return total_mcc



def main():
    """
                                            np.linspace(5, 30, num=5,
                                                dtype=np.dtype(int)).tolist(),
    """
    tuned_params = {"zscore_norm_lag" : [5, 10, 30, 60, 120],
                    "zscore_norm_threshold" : np.linspace(1.5, 2.5, num=5).tolist()}

    print('\n{}\n'.format(tuned_params))

    # create a new directory to host all re-runs
    og_prefix = 'param_opt'
    prefix = og_prefix
    i = 0
    while(os.path.exists(prefix)):
        prefix = og_prefix + '_{0}'.format(i)
        i = i + 1
    os.mkdir(prefix)

    gs = GridSearchCV(D3E(prefix_in=prefix), tuned_params,
                      n_jobs=-1, pre_dispatch=10,
                      cv=2)
    gs.fit(np.arange(200).tolist(), np.append(np.ones(100), np.zeros(100)).tolist())

    # print results and save to file
```

```
    with open("{}/log.txt".format(prefix), "a") as f:
        print('Best Score: {}'.format(gs.best_score_), file=f)
        print('Best Parameters:\n\t {}'.format(gs.best_params_), file=f)
        print('Results:\n\t {}'.format(gs.cv_results_), file=f)

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

```
from moviepy.editor import VideoFileClip

# in seconds
clip = VideoFileClip("../data/video/my_video-2.mkv", audio=False).\
           subclip(69,83)

clip.write_videofile("D2_C2_5-11-2019_BHS_CCtruck.mp4")
```

## C.4 Python Libraries

termcolor 1.1.0 pyfiglet 0.8.post1 scipy 1.2.1 dlib 19.17.0 tsfresh 0.12.0 edgetpu 1.9.2 numpy 1.16.2 moviepy 1.0.0 matplotlib 3.0.3 pandas 0.25.0 opencv_python 4.1.0.25 Pillow 6.1.0 scikit_learn 0.21.3 pyyaml 5.1.2