https://doi.org/10.2352/ISSN.2470-1173.2017.9.IRIACV-270

Traffic Camera Dangerous Driver Detection (TCD3)

Vidur Prasad; University of Michigan; Ann Arbor, MI

Abstract

The goal of the TCD3 project is to identify anomalous and dangerous driving patterns from traffic camera feeds. Successful execution can improve road safety by assisting law enforcement catch dangerous drivers, who text while driving or drink and drive. TCD3—in real time—uses Computer Vision to detect cars on the road, utilizes Machine Learning algorithms to identify cars exhibiting dangerous behaviors, and then notifies law enforcement of suspicious vehicles. The project overcomes several technical challenges such as detecting vehicles under different lighting conditions, tracking vehicles in different frames, and distinguishing random variations in a vehicle's path due to normal driving from anomalous variations due to distracted driving. TCD3's C++ script runs on a server and receives live streaming traffic camera feed. A heuristic Computer Vision algorithm utilizes optical flow analysis, background subtraction, and feature extraction algorithms to reliably determine vehicle positions. A proprietary recursive matrix density-based method was created to clean sensor feeds, sizably improving detection accuracy, and greatly improving on current morphological methods. Image registration allows a vehicle's path to be analyzed through multiple frames. A test suite of traffic camera footage was used to evaluate vehicle detection. Frames were doctored and drunk drivers were simulated to test the Machine Learning system, the algorithm was found to have an 83% accuracy. Machine Learning was used for historical and active comparative analyses of vehicle paths to identify anomaly. The system is contextually aware and is robust with respect to normal irregularities in traffic patterns such as from red lights. Permission for large scale testing of the prototype on actual high fidelity traffic camera footage has been requested. Upon detection, the relevant video clip will be extracted and sent to law enforcement for further action. To increase affordability, processing speed, and scalability, a multi-node networked Spark-based supercomputing architecture is being investigated. TCD3 is multi-threaded for maximum resource allocation. The project website is at drunkdriverdetection.com.

Introduction

The TCD3 project was conceptualized from the growing need to have automated active analyses of drivers on roads to detect distracted or drunk drivers. The aim of the project was to use open source, low-level languages to be create Computer Vision and machine learning algorithms to actively analyze traffic camera footage to identify drunk or distracted drivers. This paper will outline the different foundational algorithms that were used, the reasoning and justification for their selection, there importance, and their implementation.

Drunk and Distracted Driving

Drunk and distracted driving is a problem that is growing in our society, and is a problem that affects us all. The current paradigm for solving this pressing problem is fundamentally outdated: convince drivers to not drive drunk or distracted, or use human resources to identify these drivers. Unfortunately, this solution is flawed as it has been concretely proven that drunk and distracted driving is not on a decline, even with the advent of commercials urging people to drive safely, and with recent law enforcement budget cuts, it is becoming more difficult to deploy human assets to observe and find these drivers.

The current push to use technology to solve the problem is also flawed for two reasons, one, it requires new hardware and investments to pay for it, and two, many of these methods call for drivers to manually opt-in to self-monitor, which is not a permanent nor a viable solution.

There are over 600,000 distracted drivers on the road on any daylight hour in the United States. There are also over, on average, 300,000 injuries and deaths resulting from just drunk driving every year.

After analyzing this problem, it is clear that the only long-term, and financially, as well as technically, feasible solution is to use existing technology and have a pervasive and active monitoring system.

Technical Goal

The goal of this project is to create a hardware and software package that is able to take in live streams of traffic camera footage and process these streams to determine the position of the cars in the feed. The cars' positions are then intelligently analyzed to determine if the cars are exhibiting anomalous behavior commonly shown by distracted or drunk drivers. If a car is flagged as being suspected of being driven by a drunk or distracted driver, then the clip of video containing the car, and its license plate, is sent to law enforcement for final validation, and for further investigation.

This system will help law enforcement target its efforts by leveraging the current visual sensing infrastructure to identify suspicious drivers.

Previous Work

TCD3 was initially created using Octave, a language based on MATLAB, by Mathworks, and used simple blob analysis and manual background subtraction to identify cars. Morphological operators were used to refine detects, and the grassfire algorithm was utilized to identify and register discrete objects. A system to record the x and y displacement of the coordinates was created to perform rudimentary threshold to determine when the cars were moving outside of the normal zone. This work was instrumental in understanding the complexity of Computer Vision, and developing a paradigm that is used in the current iteration of TCD3.

Language

C++, long considered the main language to be used for Computer Vision, alongside Python, was chosen for use for its broad support with various Machine Learning and Computer Vision APIs, as well as for its low-level access to optimize systems. TCD3 is written in C++ as OpenCV, the primary library used, is natively written in C++, and other versions, such as Python, operate with wrappers. C++ offered some obstacles in

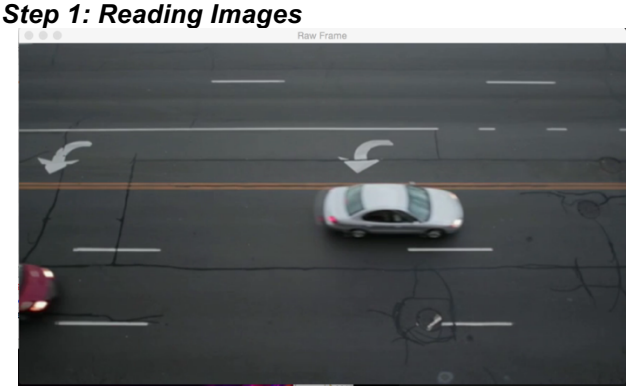
regards to memory leaks when working with larges sets of frames, but efficient use of pointers and careful memory checking was able to reduce RAM usage to approximately 2 GB.

OpenCV is a library that contains more than 2500 algorithms for use in Computer Vision and Machine learning. OpenCV provides a Matrix data structure that has broad support across multiple other libraries, such as BGSLibrary. OpenCV, as it is completely implemented in C++, allows all of its algorithms to be edited and tweaked for the specific use case.

Phase 1: Computer Vision

Stage 1: Generate Background Model

Step 1: Reading Images



TCD3 reads video files, of standard MP4 or MOV type, and converts the files into individual frame that are then passed through the system to be processed.

This architecture allows all frames to be stored once, reducing RAM load, and allows all frames to be saved only once, reducing IO usage. Client methods can additionally clone data from global vector if analysis will edit actual frames. On startup, a buffer of frames are loaded into the vector, usually around 100, and the architecture is in place to allow methods to access the buffer as it is being built to begin initializing their systems.

The buffer is also created to ensure that no systems ever cause an out of bounds error, as there is sufficient memory. The size of the buffer controls multiple events as it effects the time it takes to train, and effects the timing of initialization as many steps must be completed in sequence, as they cannot be done in parallel, and proper relational timing is kept to ensure that no matter the size of the buffer memory, there are no runtime errors. The size of the buffer, also, as expected, affects the quality of the Background Detection, as many operations, such as the median background generator, rely on large data sets to reliably estimate and separate the background from the foreground objects. An additional vector of gray scale frames is also maintained, as certain algorithms do not require color. The raw frames that are read and kept for conversion are deleted at the end of every cycle, after one frame has been processed, and are scrubbed to reduce RAM usage and ensure that there are not memory leaks.

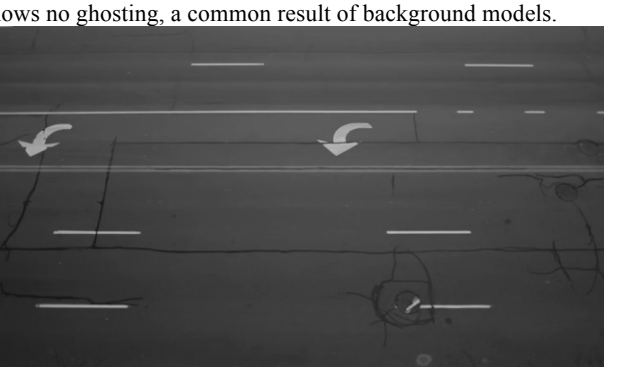
Step 2: Background Model Median

The first background subtraction method that is used is the median model, or BMM. TCD3 can generate BMMs, and can read a BMM, stored as an image and containing the median value for each pixel, and perform analysis. A BMM is generated by stepping through the entire buffer memory and creating a vector of integers of every pixel value in history for every specific position, and calculating the median value.

A mean based model was investigated, but an accurate reading required a very large buffer memory, and removing ghosting is impossible, as variations will constantly affect the learned average. Another pro to using a median based model is that with advanced sorting methods such as Tim Sort, a median calculation is highly efficient.

The BMM generation runs on its own thread, and has the ability to perform simultaneous operations by separate the image and process using multiple threads to decrease compute time. As the current testing rig has only 8 cores, further branching of threads will not increase speed, but it is an option for the future. The BMM model is generated every 1.5 minutes to account for changing weather conditions, as well as general lighting changes. The thread silently works in the background, and does not slow runtime as the new model is used only when the BMM is successfully generated.

A sample of a BMM generated model is shown below, at no point in the test footage was a picture of the road without cars ever shown, proving that the BMM model is able to effectively generate a background model. The BMM model shown was generated with a buffer memory of 100 frames. The image also shows no ghosting, a common result of background models.



All BMM models are generated in gray scale for speed, as a RGB model would require 3X time, and as background subtraction is more efficient and clean using gray scale images.

Step 3: Gaussian Model (Kaewtrakulpong & Bowden)

Multiple background subtraction methods exist that use techniques such as BMM, however, these methods, especially BMM, suffer as a result of lighting changes and are forced to recalculate a model periodically to remove these changes.

The Kaewtrakulpong & Bowden approach is designed to use a Gaussian Mixture Model in conjunction with revised model update equations to allow for more accurate lighting adjustment, as well as to allow for shadow removal in the model. The approach, referred to in the OpenCV library as the MOG1 approach, utilizes this approach to reduce the training time to detect the objects.

TCD3 utilizes MOG1 and trains during run time by feeding color images to fill the GMM. This model also uses a kernel estimator for each pixel that allows regular movements, such as in trees in the image, to be ignored. Other systems utilized by TCD3, such as Optical Flow Analysis, are also used to further crosscheck the models, and increase overall accuracy. MOG1 runs on its own separate thread, and is one of the slowest of the Computer Vision algorithms used by TCD3 for real time computation.

Step 4: Gaussian Model (Zivkovic)

The second Gaussian approach utilizes recursive equations to actively update the parameters of the model, such as the number of Gaussian mixture models, and size of the memory for each pixel. The Zivkovic approach is referred to as MOG2 in OpenCV. This adaptive method offers better response to objects such as wildlife or shadows that normally mess up other subtraction methods.

The primary purpose of running both the Kaewtrakulpong & Bowden and the Zivkovic approach was to adapt the model to the conditions, allowing for a system that is able to update and reflect

the scene as accurately as possible. Running both models also allows multiple guesses to be made, which increases accuracy.

This model furthers the paradigm that the system should be able to adapt to any situation with minimal initial input, as it is able to adapt to lighting and weather changes without requiring outside sensor input, and increase the chance of a successful detect. MOG2 runs on its own separate thread, and is one of the fastest methods for real time computation.

Step 5: Gaussian Mixture Model (GMM)

TCD3 also uses a traditional GMM model, using three mixtures, to adjust and deal with changes in the lighting conditions. The traditional GMM model does not work as well as MOG1 or MOG2 as it is not able to adapt to changing conditions as well, and is also not able to intelligently develop a GMM.

After multiple tests, it was determined that using a three mixture model gave the most accurate detect, but due to the variance in the number of mixtures actually found in the sensor data, the traditional GMM model has many false-positives. To help reverse this, the buffer memory for the GMM is tripled.

Of the detection methods, the traditional GMM is weighted the lowest in the final confidence detector, and future versions will quite likely stop using the traditional GMM and rely solely on MOG1 and MOG2 for a GMM based background mode.

Step 6: Background Model ViBe

ViBe is a commercial background subtraction algorithm designed to perform analysis over multiple frames to create a background model. ViBe works by analyzing the values of a pixel over a period and then calculating the probability density function. This method cause ViBe to require a long training time, about seven times the buffer memory, to get a proper understanding of the background model. Through its use of advanced statistical analysis of pixel histories, ViBe creates a highly accurate model of the background. Given enough time, ViBe returns a model of the cars better than both MOG1 & MOG2. TCD3 runs ViBe even after real-time analysis begins, and continues training ViBe after an appropriate training period, about seven times the buffer memory.

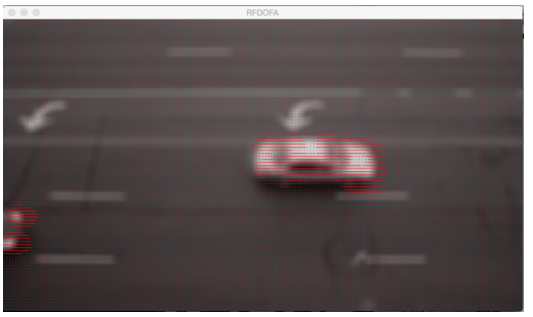
Stage 2: Detect Cars

Step 1: Background Subtraction

The next step in TCD3 is to use the generated background models to perform subtraction with the current frame to determine the difference, or objects in the frame.

Background subtraction starts with the Median background frame, which is subtracted from the raw gray scale frame. This returns a Matrix where the amplitude of the difference correlates with the actual difference, allowing for thresholds to be applied. After subtraction, a binary image is generated with every value above 50, eliminating small differences due to shadows, saved as one. Using a binary image also greatly decreases compute time. The various models also generate binary masks that show the difference, calculated through various methods, to identify objects.

Step 2: Optical Flow Analysis Detection

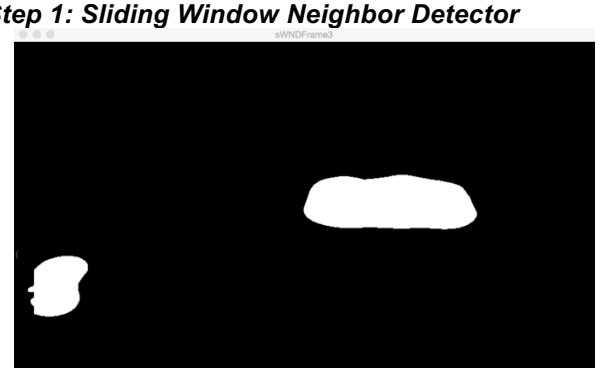


During the planning stages of TCD3, research work was conducted to gain an understanding how humans understand images. One of the main things that was gleaned from this research was that humans use a multi-faceted approach, using different “Computer Vision” like methods to create a composite model, from which a final decision to recognize an object is done.

To emulate this method, TCD3 utilizes multiple background subtraction methods to understand an image. However, to add redundancy, an Optical Flow based method was created to detect objects. The Farneback Dense Optical Flow model was used to generate a model, for each pixel, that showed its movement frame over frame. Using this Optical Flow model, a threshold is run to detect every pixel that has movement. From this, the Sliding Window Neighbor Detector is used to clean the image, and develop a complete model of all large motion in the image. This is an effective method as all the cars, no matter how optically complex and similar to the background, can be caught through motion. The Farneback Optical Flow algorithm creates an image of motion vectors, from which a heat map of motion is generated.

Stage 3: Clean Data

Step 1: Sliding Window Neighbor Detector



Various morphological approaches were tried to attempt to repair holes and other issues with the results of background subtraction, but required too much compute time, as well as did not do a good job. One of the biggest issues was that running repeated morphological operators degraded the quality of the detect by causing multiple cars close together to morph into one, as well as were not adequate at removing noise.

To address this problem, a custom recursive algorithm was created to read in raw background subtracted day and perform image cleaning in a manner reminiscent of morphological operators. The Sliding Window Neighbor Detector, or SWND, is designed to determine a pixels value based on the density of its neighbors, and therefore fill holes and remove noise, while retaining the quality of an outline of a well define object. The algorithm works by sliding through each pixel while observing the area around the pixel to determine if the pixel is part of well-defined object.

A pixel will be determined as existing if over 25% of its neighbors exist. This ensures that holes inside of images will be filled, as they will have non-adjacent relatively close neighbors that cause the SWND to detect a pixel. SWND is also able to remove noise, as it is able to identify a pixel as being a loner as its neighborhood does not contain many other pixels.

The property of SWND that makes it powerful, and quite accurate, is its ability to work recursively by starting with relatively small sectors to larger and larger sectors that carefully remove more noise, fill larger holes, and, more importantly, careful knead the objects into cleaner, more ovular shapes. This approach drastically reduces noise, and improves object detection.

The SWND is usually run three times with quadrupling sector size to get a proper detect. SWND is used for all data

sources, and is one of the primary innovations in the Computer Vision Phase as it allows for all sensor values to be carefully analyzed, and equalized as they all pass through the same processer. Equalization is critical as a single car must keep a similar detect through its path to ensure that all deviations are caused by motion, not by inaccurate detects. SWND is optimized to move through the image in the most efficient path as possible, and does the minimum movement. SWND also attempts to scrub temporary variables as quickly as possible to reduce RAM usage. As SWND is imperative for a detect to be considered finished, it does not open a new thread, and instead is launched from each thread running computer vision.

Step 2: Canny Contour Detector

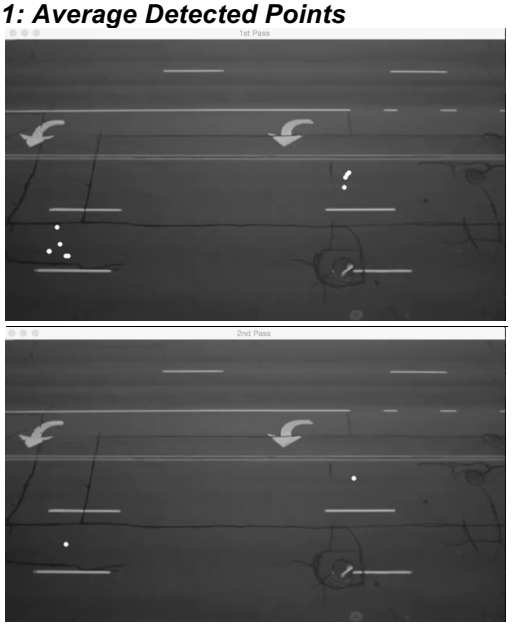
The next step in the process is to take the raw binary images and determine the center point of all of the cars. To detect the individual cars as discrete objects, the Canny contour detector is used to identify just the outlines. A raw threshold is then used to remove objects too small to be a car. Objects that are on the outsides, where a complete detect is not yet possible, are thrown out. This is to ensure that all of the anomaly detection happens after a proper solid detect is acquired.

After the canny contours are detected, the center points of the contours are saved into a global vector of points. This method allows every car detected, through each of the different Computer Vision methods, to be saved in one vector.

Phase 2: Tracking Machine Learning

Stage 1: Process Coordinates

Step 1: Average Detected Points

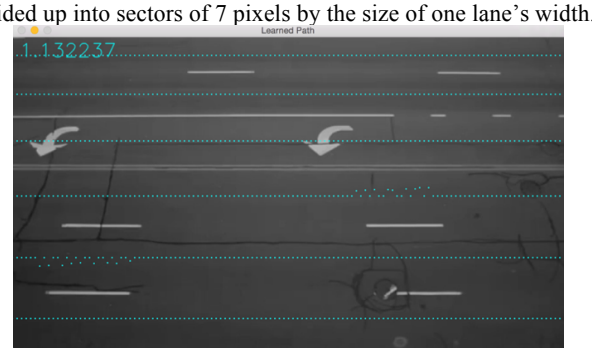


After all of the different Computer Vision methods run, a method to cluster the points was devised. K-Means is the optimal method, but it requires that one knows the number of cars before running the clustering algorithm.

The averaging system works by identifying all the points close to each other, and then averaging them and finding the center point of the cars. The distance threshold is set so that it is large enough to detect all points that are within a car, but small enough to not detect two adjacent cars. The final detected points are then saved into the same vector of detected points.

Step 2: Learning Area Sector Model (LASM)

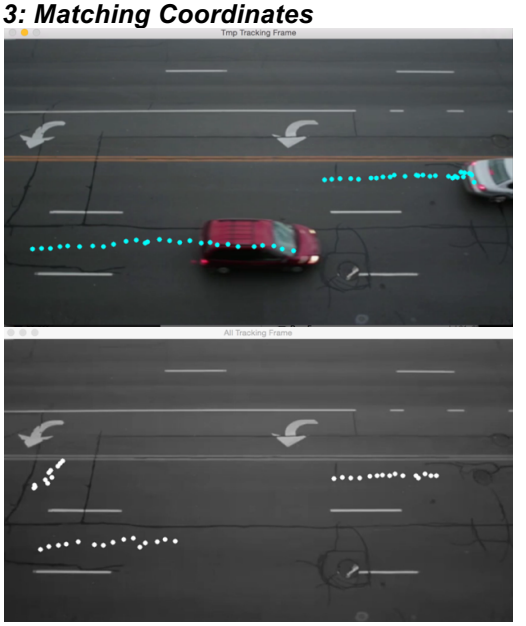
One of the central qualities of TCD3 is the ability to be able to adapt and work in a diverse array of highway situations. One of the models used is an area sector model, where an image is divided up into sectors of 7 pixels by the size of one lane's width.



This requires the number of lanes to be entered into the system. A grid is created where a car is classified into the lane that it is traveling in, and then every general area, 7 pixels across, to have an updated model of where most cars are when they travel through each sector. This allows the model to continuously update and become more accurate the longer it is used. This model is generated so that thresholds can be applied to determine if a car is outside of the safe range. This model is useful so that scenes with multiple different roads and different directions and speeds will be able to intelligently apply correct models to appropriate cars.

LASM will eventually be modified to include angular and movement data, in addition to position data, for each of the individual sectors. The LASM is useful as it is an infrastructure that is able to utilize any metric and save it into sectors. LASM, and LASM Enforcement, are designed to use raw thresholds, as well as learned models. LASM & LASME are generated using a vector of positions that is entered by the user, which shows the boundaries between each lane, and run on the main thread.

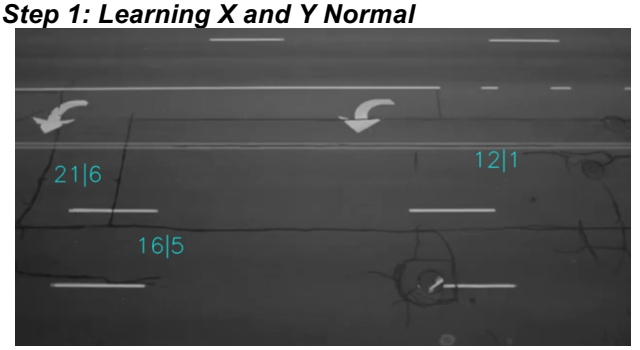
Step 3: Matching Coordinates



The next step, prior to running deeper analysis, is to determine the x and y displacement for each car. This is done by saving all of the coordinates of the cars into a vector of vectors of coordinates, and the comparing the current frame of detects to the previous frame of detects. A matching algorithm was created to allow cars to be tracked frame over frame, and from this, the x and y displacement of each car was extrapolated.

Stage 2: Movement Property Detection

Step 1: Learning X and Y Normal



The first machine-learning algorithm that is run is a simple averaging system that accumulates the x & y movements for every car, and then average by the number of detects, thereby building an accurate model of the movement. The x & y movements are useful as they can be used to identify excessive movement, as well as changes in acceleration, which is a main sign of drunk or distracted driving. The movement property calculations are all calculated on the main thread, after Computer Vision.

Step 2: Learning Angular Movement

The second machine-learning algorithm is to determine the normal range of angular movement that cars exhibit. The angular movement is calculated by computing the inverse tangent of the x and y movement. The angular movement is an excellent indicator to identify anomalous behavior as it gives a clear indication of the direction of acceleration, and shows if someone is swerving. The angular movement is learned so that the limit of normal movement can be learned. It is from this metric that the threshold value for anomalous behavior can be calculated.

Step 3: Learning Speed

The next step, designed to also perform the job of speed cameras, is to use the Pythagorean Theorem to calculate total movement in each frame. The normal displacement, as it is calculated frame over frame, yields the speed.

Stage 3: Detecting Anomalies

Step 1: Hard Threshold Detection

One method to detect anomalies is to use hard thresholds for each of the different properties. The hard threshold method is used to catch anomalous cars even if the learning stage has not created a reliable model of drivers yet.

The first hard threshold that is set is for the individual x and y displacement. This is useful to find users making quick swerve movements. The thresholds are calibrated based on the situation, and are applied to all of the different paths. The x and y displacement is calculated with an absolute value, so that it works for both direction of lanes. In some situations, taking the inverse of the x and y displacement is useful for some situations that have roads going in different directions. The issue with the hard thresholds, and the reason that the learning system is given a higher weight during actual run, is that it is difficult to craft thresholds that work for each of the different lanes and parts of the road. This is one of the primary reasons that LASM is used.

Step 2: Learned Model Detection

The primary method that TCD3 uses to detect anomalies is by comparing active movement with the learned model. For each of the metrics, a proportional, as well as absolute distance based method is used to recognize anomalies. For the proportional system, if the anomaly exceeds a 25% safety zone, for any of the metrics, then it is considered anomalous. However, in some cases, the safety zone may be edited to tune TCD3's sensitivity.

Any time an anomaly is detected, it is written to a frame of anomalies. This frame of anomalies is cleared after a set time, the time it takes for a car to go across the frame, after the first anomaly is detected. If more than one anomaly is detected in the same frame, a separate frame of anomalies is opened and saved to. The confidence of an anomaly detect can also be ascertained by counting the number of movement properties that exceed limits, the amount of time that the cars are anomalous, as well as the amplitude with which the car exceed safe limits.

If these frames have more than the required number of anomaly detects, usually at least 10, corresponding to 2/3 of second of anomalous behavior, at 15 FPS, the corresponding video is tagged as anomalous. This method ensures that one time detects, or small movements are not considered anomalous, and a long registered set of anomalies across the life cycle of a cars movement across the frame is detected. This method ensures that small variances are not caught, and only large deviations are caught that occur over an extended period of time. This is critical in removing false positives, and removing instances where the system incorrectly identifies one or two movements as anomalous.

A long series of anomaly detects therefore shows, with high probability that the driver is indeed anomalous. This system runs on the main thread, and is the final step in the process.

Phase 3: Processing Enhancements

Stage 1: Multi-Threading

One of the main methods used to optimize TCD3 was to multi-thread the system to spread processing load across multiple cores. The pthread library is used to open threads, check successful thread completion, and close threads after the task finished. For each of the major tasks listed above, a new thread is opened and global variables are used to pass data into the threads. All threads poll the same global vector that contains the frames.

The paradigm for processing and controlling the multiple threads is to utilize thread handlers that each handle a set of threads and report completion back to the main thread. Each of the major steps has a thread handler that launches all necessary threads to finish a step.

A more abstract thread handler then waits for its child threads to finish, before terminating completion and reporting back to the main thread, where a while loop executes until all major Computer Vision steps are complete. This system is designed specifically for scalability and to be able to add more steps to the system without increasing compute time. This system of abstraction also makes it

easier for more methods to be added, allowing the system to be continually updated, and work with the existing infrastructure.

Stage 2: Saving Models

Efforts were also made to lower compute time by reducing the amount of things that needed to be calculated every frame, and therefore allow the system to work faster. Some of the models, such as the BMM, can be saved and read during run time to avoid recalculation. This is part of the primary processing paradigm of TCD3 all steps should be executed in series if and only if there is no way to perform the steps in parallel. This paradigm has allowed the system to be optimized as much as possible, and will allow more powerful hardware to significantly decrease compute time.

Phase 4: Conclusions

Stage 1: Testing

One of the challenges to creating a system such as TCD3 is that there is a lack of footage to test and see if the system is able to detect drunk or distracted drivers. To rectify this, it was decided that drunk or distracted driving footage would be artificially generated. To do this, iMovie was used to edit in cars with tracks that emulated drunk drivers. Using the path-editing tool, a car was edited in to show it swerving through the frame, as well as speeding through the frame. After creating multiple videos, they were run through the system to ensure that TCD3 would be able to detect a variety of anomalous movements accurately. 36 different anomalous car paths were created, of which 34 were accurately flagged by the system, with an additional 4 false positives. Therefore, 30/36 tests showcased accurate results, with an accuracy rate of 83.33%.

Of all of the different anomaly detectors, the angle detector was the most accurate of the anomaly detector, as it had the ability to identify even small motions as they lead up to a large anomalous motion as it could identify a car moving toward the edge.

The system was also tested by running footage of cars changing lanes, and making small anomalous movements. This was designed to ensure that TCD3 was smart enough to avoid false positives. This system of creating test footage to ensure that the different parts of TCD3 all work effectively is critical to ensuring that it is ready to work in reality.

Stage 2: Future Work

The future of this project is very bright as there is a strong foundation currently implemented and tested. The future work focuses on facilitating faster processing by allowing for memory to be parallelized, including implementing a Spark based system. In addition, further traffic camera footage will be pursued, from a diverse set of environments, to ensure that the system retains accuracy beyond the initial test footage that was used.

Stage 3: Impact

This project will have a massive impact on the way that we deal with drunk and distracted driving, as it will finally allow us to move our enforcement of DUI laws into the 21st century. The solution that was designed is crafted for scalability, and will therefore make it possible for use in a variety of cities of different sizes. The system utilizes many algorithms designed to maximize efficiency to increase accuracy. By having a system that is able to utilize the current resources and leverage current technology to help law enforcement, we have eliminated the need to consistently use human resources to monitor roads. The system is also intrinsically private, as the license plates of cars are only recorded, and the data is only given to humans when a car's path is flagged as exhibiting anomalous behavior. In conclusion, the technical advancements that we have made throughout this project have allowed for a system that is able to efficiently deal with the pressing issue of distracted and drunk drivers.

References

[1] CDC. (n.d.). *Impaired Driving: Get the Facts*. Retrieved from Centers for Disease Control and Prevention: http://www.cdc.gov/Motorvehiclesafety/impaired_driving/impaired-drv_factsheet.html

[2] Droogenbroeck, Marc Van. "ViBe: A Disruptive Method for Background Subtraction - Van Droogenbroeck Marc." *ViBe: A Disruptive Method for Background Subtraction - Van Droogenbroeck Marc*. ViBe, n.d. Web. (n.d)

[3] Foschi, Patricia G. "Feature Extraction for Content-Based Image Retrieval." (n.d.): n. pag. *Feature Extraction for Image Mining*. San Francisco State University. Web

[4] Kokash, N. *An Introduction to Heuristic Algorithms.* PSU.

[5] Jain. "Edge Detection." (n.d.): n. pag. *Edge Detection*. University of Nevada Reno. Web.

[6] *Morphology Fundamentals: Dilation and Erosion.* Mathworks.

[7] Reynolds, Douglas. "Gaussian Mixture Models." (n.d.): n. pag. Print.

[8] Shi, Jianbo, and Carlo Tomasi. "Good Features to Track." *Good Features to Track* (1994): n. pag. Web.

[9] Shrivakshan, G. T. "A Comparison of Various Edge Detection Techniques Used in Image Processing."*International Journal of Computer Science Issues* 1st ser. 9.5 (2012): 269-76. *A Comparison of Various Edge Detection Techniques Used in Image Processing*. International Journal of Computer Science Issues, 1 Sept. 2012. Web.

[10] Sun, Min, and Srdjan Krstic. *Computer VisionToday's Outline* (n.d.): n. pag. *Optical Flow*. Princeton. Web.

Author Biography

Vidur Prasad is a student at the University of Michigan pursuing a dual major in Computer Science and Economics. He has worked at the University of Dayton Research Institute and at Aptima, Inc. on a variety of Computer Vision and Machine Learning applications. He has presented his work at the Intel International Science and Engineering Fair. Moving forward, he would like to continue his work in data analytics through a career in Management Consulting.