TMA (Truck Mounted Attenuators) alert system-

Development and Testing

A Thesis

Presented to

The Faculty of the Graduate School

At the University of Missouri-Columbia

by

Mohammadmehdi Zoghifard

Dr. Yaw Adu-Gyamfi, Thesis Supervisor

July 2022

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled

TMA (Truck Mounted Attenuators) alert system-Development and Testing

Presented by Mohammadmehdi Zoghifard,

A candidate for the degree of

Master of Science in Civil Engineering

And hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Yaw Adu-Gyamfi

Dr. Timothy Matisziw

Dr. Sabreena Anowar

ACKNOWLEDGEMENTS

I would first like to thank to my academic adviser Dr. Yaw and my co-advisor Dr. Carlos Sun, whose encouragement, suggestions, and support helped me overcome many difficulties in my research, study, and life.

I would like to thank Dr. Mark which it wasn't with his help, this research could have taken longer. He answered so many of my questions with patience and put time into discussing about this research.

A warmest thank you to my parents whose encouragement, emotional support and prayers, made me successful in my entire life. I couldn't have done it without them. They have fostered confidence and passion in me to work hard and pursue my dreams with no boundaries.

And thanks GOD. Without him, I wouldn't be here. God has helped me through so many struggles and answered so many of my prayers.

Contents

Table of Figures:

Table of Tables:

Abstract:

Truck Mounted Attenuators (TMAs) play a crucial role in safety of work zones as they decrease the impact of the crashes, reduce fatalities and injuries, and increase safety. However, there are almost no solid solutions to decrease the number of crashes with the TMA truck while maintaining the safety of the work zone workers. In this study, we aim to alarm the drivers following the TMA truck to avoid collisions and consequently, decrease the number and severity of the crashes. We used Unity 3D as the simulator to create scenarios with a smooth 45-degree and steeper 90-degree curves. Then we ran the simulator with different vehicles volumes and speed, including aggressive drivers in the simulator, and overall, creating different crash scenarios involving TMA trucks making some scenarios that some cars crash with the TMA truck which are rare in real life. Furthermore, computer vision has been used to define a safety zone on simulator videos to automate triggering the alarm when necessary to avoid crashes when vehicles cross the safety zone boundaries on the same lane as the TMA truck. After that, we used field videos from a TMA truck to evaluate our system. Results show that the proposed system achieved an average accuracy of 76.6% and 65% in simulator videos and TMA field video respectively. The only downside is having a fixed safety zone which causes problems when the geometry of the road changes or the TMA truck rotates to some degree and causes false alarms for the

vehicles passing in the other lane. Overall, this system showed promising results and can be implemented in real-time for the TMAs to reduce collisions.

Topic: TMA (Truck Mounted Attenuators) alert system-

Development and Testing

CHAPTER 1 INTRODUCTION

Work zone safety is a crucial matter in transportation. Work zones can be stationary or mobile. In this study we focus more on mobile work zone, however, the method that is presented in this study is applicable to stationary work zones as well. The major contributing factors to traffic accidents are driver inattentiveness, fatigue, and immature behavior. Nearly 25% of police-reported crashes, according to the National Highway Traffic Safety Administration (NHTSA), involve a driver who was either preoccupied, tired, sleepy, or "lost in thought." About half of the crashes that involve inattentiveness are because of driver distraction. Driver distraction includes visual, cognitive, biomechanical and auditory distraction (1). The second major factor, fatigue has a contribution of more than 25% in road crashes (2). It was found that human behavior is the main cause for the crashes. The goal of this study is to automate an alert system using computer vision for TMA trucks to alert the drivers who get close to the TMA truck to avoid crashes.

There have been some studies that investigated work zone crashes and introduced new methods such as Automated Flagger Assistance Devices (AFADs) (Figure 1) that are implemented in stationary work zones to increase safety in the work zone. Green light is another example that is being implemented on the trucks such as a snow-removal truck (Figure 2).

Figure 1. An image of an AFAD (Safety Technologies 2015b) (3)

Figure 2. ODOT green light snow-removal truck (ODOT 2013) (4)

A truck-mounted attenuator (TMA) (figure 3) is a tool that mounts on the back of a working truck to protect workers and the running vehicles from the rear-end collisions between vehicles and a slow-moving truck. Their purpose is to minimize property damage and save lives. TMAs are designed to absorb most of the impacts both in low-speed and high-speed moving vehicles. Figures 4 and 5 are examples of vehicle collisions to the TMA in both high speed and low speed respectively:

Figure 3. Truck Mounted Attenuator (TMA) (5)

TMAs are designed to absorb most of the impacts both in low-speed and high-speed moving vehicles. Figures 4 and 5 are examples of vehicle collisions to the TMA in both high speed and low speed respectively:

Figure 4. Crash with a TMA at high speed (5)

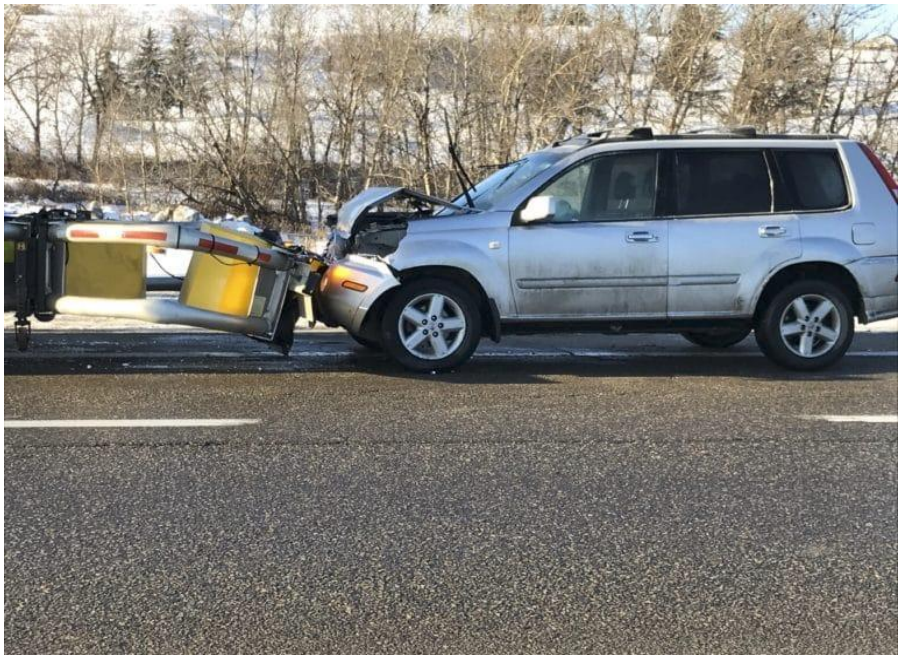

Figure 5. Crash with a TMA at low speed (5)

Mobile work zones move slower than the traffic which can surprise distracted drivers. With distracted driving increasing, there is a need to alarm the drivers

when approaching work zones, so they won't crash with the work zones. Implementing an alarm system would decrease fatalities and injuries for both workers and general traffic. Although TMAs are absorbing the impacts of these collisions and can save a lot of lives and money, there is a need to decrease TMA crashes. The speed difference between the moving vehicles and the moving TMA truck is usually substantial, and that is why there are numerous collisions happening with the TMA. But what if we could minimize the number of crashes with the TMAs?

In the past, all the audible alert systems for the TMAs were operating manually. Meaning that a work zone worker on the back of a TMA truck was triggering the alarm. The issue is that, at the time of crash with the TMA, the worker's life is in danger. Moreover, the worker can get distracted or get tired after a period. Thereby, automation can become handy and replace this manual process. Automation and machine learning models in areas such as smart cities, autonomous cars, healthcare, etc., have significantly improved as a result of recent developments in deep learning. With emerging technologies, everything is becoming more efficient. There are myriad studies about the use of automation in transportation. Vehicle detection for collision avoidance systems (6), adaptive cruise control (7), (8) and (9) , automatic braking system (10), lane keeping (11), (12), autonomous driving (13), crack detection in pavements (14), to name but a few are examples of using automation in vehicles or transportation studies.

Automation makes tasks easier and more efficient and eliminates the human errors. In order to eliminate the presence of a worker on the back of a TMA, we need to automate this process. The automation in the TMA has the potential to solve this problem. With automation, we can protect workers and reduce injuries. In this study, a system for automation in TMA has been provided.

This study has been done using mixed simulator and computer vision. The objectives of this study are to develop a simulator with Unity 3D to simulate a TMA driving environment and conditions for the rest of this research. Then develop an artificial intelligence system to implement on the simulator to detect vehicles and trigger the alarm when they enter a safety zone created with computer vision to alarm the incoming vehicles. After improving and evaluating this system on the simulator, we will test this system on the TMA field videos which was taken from back of a TMA. Then we will evaluate the system and results by watching the recorded videos to see how many false alarms and true alarms we get from our system.

Why use a simulator?

Since there are not a lot of videos from the back of TMAs, especially videos of TMA crashes, the simulator has been used to record these scenarios and then analyze with computer vision. Furthermore, we need to use simulator because of availability of a TMA and not having a lot of field videos.

CHAPTER 2 LITERATURE REVIEW

The use of the TMA has been on the rise because of the safety that it provides to the workers in the work zones. There are some studies that talk about crashes that are happening with the TMA which are described in the following paragraphs.

A study shows that in Missouri between 2012 to 2017, there were 139 TMA crashes. Among the contributing factors, distracted driving with 44 percent was the main reason for TMA crashes. 70% of these crashes were with mobile work zones and the remaining were with stationary work zones. Factors involved in TMA crashes and type of activities when these crashes happened are brought in table 1 and table 2 respectively (15):

Table 1. Factors involving TMA crashes in Missouri (15)

	Distracted driving	**Late merging**	**Speeding**	**Others**	**Not reported**
Count	61	21	7	7	43
Percentage	43.9%	15.1%	5.0%	5.0%	30.9%

Table 2. Mobile work activity at time of TMA crash in Missouri (15)

Operation	**Count**	**Percentage**
Pothole patching	31	22.3%
Striping	20	14.4%
Sweeping	18	12.9%

Maintenance (not specified)	13	9.4%
Bridge	9	6.5%
Mowing	8	5.8%
Cleaning dirt	8	5.8%
Signage	7	5.0%
Spraying weeds	3	2.2%
Rolling	2	1.4%
Other	9	6.5%
Unknown	11	7.9%

In 2020 in the U.S., there were 857 work zone traffic fatalities in which 20% of that number involved rear-end collisions. In 2019, this ratio was 24% meaning that in this year, there were more rear-end collisions. Also, data suggests that speeding was 5% more as a factor in work zone crashes in 2020 (16) . This is also in line with our Covid-19 impact analysis on transportation. Our finding suggests that due to the decrease in traffic volume in 2020, drivers were speeding more, causing more fatal crashes although overall crashes were reduced by 16.8% compared to 2019. Overall, in 2020, total fatal crashes had increased.

Table 3. Percentage increase of crashes 2019-2020

Year	Total Fatal Crashes	Fatal crashes in work zones
2020 over 2019	12% increase	1.4% increase

In a study, it was shown that 92.2% of the TMA crashes were rear-end crashes, which shows the importance of the TMA crashes in a mobile work zone. After analyzing the involving factors in mobile work zone crashes, we found that about 39% of the crashes involved lane changing (17). MoDOT owns more than 300 TMAs in Missouri and reported that year 2021 with 61 TMA crashes was the highest crashes in the 4-year report from 2018 to 2021 (18). A study shows that more than 80% of the crashes were the fault of the third parties and among the contributing factors, 66% of those crashes were caused by distraction. On average, each TMA-cars crashes saved over $196,000 in crash costs and demonstrated that TMAs were highly effective in decreasing the severity and cost of the crashes (19). In another study that investigated the contributing factors of TMA involving crashes, it was found that about 8% of the crashes were because of fatigue and falling asleep. More than 87% didn't involve vision obstruction by the TMAs and about 12% had a possible vision block such as glare, blind spot, or moving vehicle. Most crashes occurred with passenger vehicles and then trucks are the second most category (20).

In a new approach for TMA-involved crashes based on existing work zone crash prediction models, in the state of Missouri from 2011 to 2016, it was found that from 181 analyzed situations, 97 crashes were in normal conditions (more than a half), 66 crashes were in the categories of absolutely safe, very safe, and relatively safe conditions. These categories with normal conditions were covering

90% of the crashes, and on the other hand, about 10% were unsafe conditions. Moreover, it was found that 6 of the 12 unsafe conditions, were very unsafe which half belonged to roadside work. 7 out of 12 unsafe conditions were belonged to other work which needs to attract more attention. (17).

A study evaluated field tests of alarm systems for mobile work zones, particularly an alarm device and a directional audio system. They have tested three operation modes such as continuous, manual, and actuated. Figure 6 shows a construction vehicle with TMA for an alarm device and a directional audio system. It was found that horizontal and vertical curves and TMA truck movement caused false alarms and false negatives. They have demonstrated that mobile work zone alarms can increase safety. The results from the test shows that in all warning setups, except actuated setup, average merging distance increased. Directional audio system was the highest by 122 ft and decreased the average vehicle speed by 3mph. Increasing the average merging distance and decreasing the average vehicle speed will increase safety (21).

Figure 6. Construction vehicle with TMA for (left) an alarm device (right) directional audio system (21)

It was found that horizontal and vertical curves and TMA truck movement caused false alarms and false negatives. They have demonstrated that mobile work zone alarms can increase safety. The results from the test shows that in all warning setups, except actuated setup, average merging distance increased. Directional audio system was the highest by 122 ft and decreased the average vehicle speed by 3mph. Increasing the average merging distance and decreasing the average vehicle speed will increase safety (21).

A study about work zone workers perception of automated TMAs, shows that they are overall positive and accept this technology and expect that this would reduce crash severity. However, workers are concerned about trusting this technology under abnormal conditions such as poor visibility and higher traffic

volumes. Those who were trained more for automation in TMA and had more experience, had more trust. (22) .

Numerous studies have been done over vehicle detection. There are various vehicle detection techniques. Two types of techniques are motion-based and appearance-based (6). For the motion-based, there is optical flow technique with detection rate of 60% for 5km/h and 90% for 25km/h relative speeds which is calculated by matching pixels between two consecutive frames. Optical flow is not accurate (60%) at lower relative speed and has high computational cost and is sensitive to the camera movement (23). From the appearance-based, there is Shadow Underneath Vehicle technique which uses image thresholding approach, and it detects easily with less computational costs. However, it fails when pavement color is uneven, and it has problems with long shadows in different times of day and it gets affected by the near building shadows (24). Corners technique clusters based on corner coordinates and types, but it fails in complex environments (25). Vertical and Horizontal Edges technique which is the ration of horizontal to vertical edges for vehicles is fixed and it has easy computation. Its limitation comes to interference from outlier edges, and it is also hard to choose an optimum threshold (26). In Symmetry technique, the vehicles front and/or rear views are symmetrical vertically and has good ROI estimation but has high computational cost and needs a tough estimate of vehicle location in an image (27). Color technique, which is a simple histogram calculation, is also good in

nighttime detection. But when the background color is the same as the vehicle, the performance drops and it is highly dependent on illumination (28). Vehicle's Lights technique, which is only for nighttime detection, will extract bright objects, however, it can be confused with other lights on the street (29). Stereo Cameras technique, which computes a disparity map in order to get a 3D map and it is good for detection of small size objects. However, it is expensive and has high computational cost (30). "Multiple features" technique has a better localization and high precision rate, but it is computationally expensive and time consuming (31).

CHAPTER 3 DEVELOPMENT OF METHODOLOGY

Research framework:

The framework of this research is being divided into two main components, simulator development and computer vision development. Below, the overall framework of this research is provided.

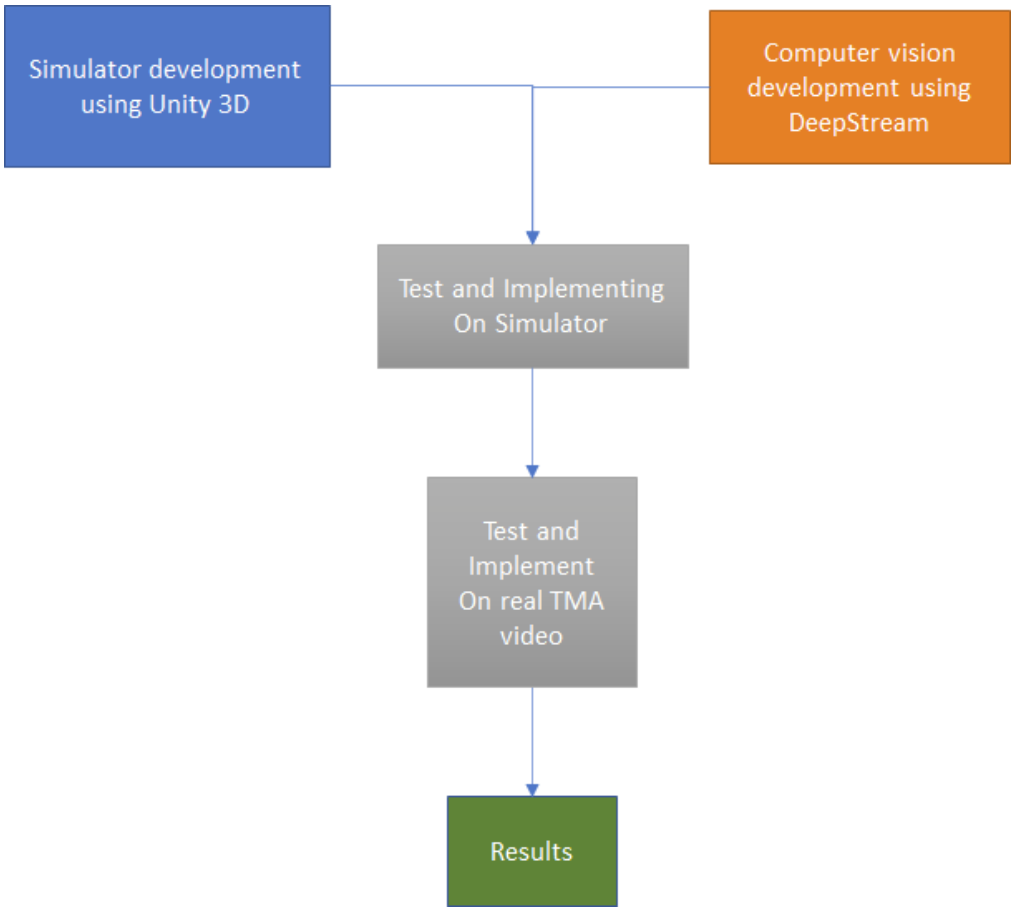

Figure 7. Framework of this research

3.1 Simulator Development

Developing the simulator had several steps. The first step was to create roads. Two separate roads have been created; both are straight roads except that there is a curve 3-4 seconds after the simulation starts. One with a 45-degree curve, and one with a 90-degree curve. Both roads are about 10 km long and the simulation stops after all the vehicles reach the end of the road. These scenarios were built with RoadRunner by Mathworks. RoadRunner is an interactive editor that lets users create 3D scenes for simulation. The other option was to use Unity 3D itself which is our main simulator platform for this study. Using RoadRunner to build the environment had a big advantage in terms of speed and simplicity comparing to Unity 3D. I created the scenes using RoadRunner and then exported it to Unity to continue the rest of the development.

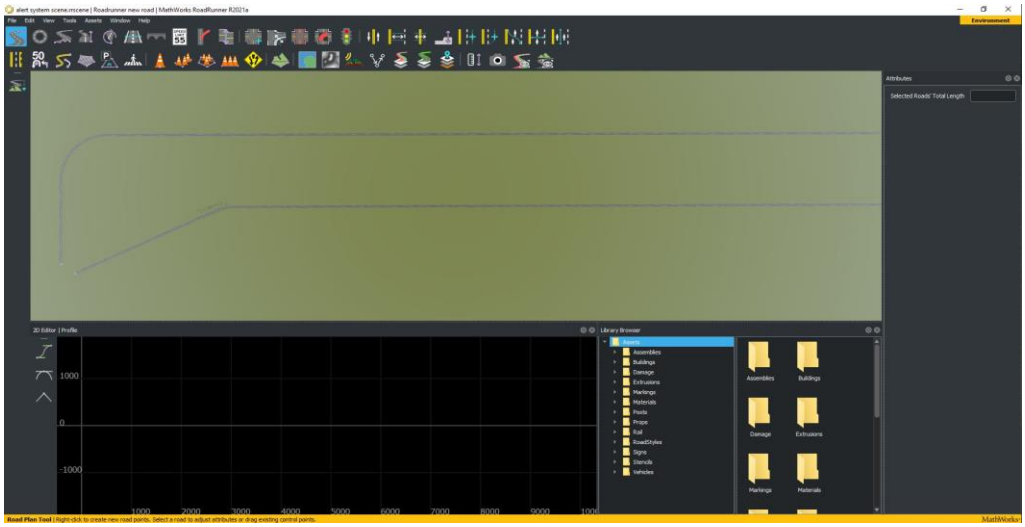

Figure 8. Both roads in RoadRunner

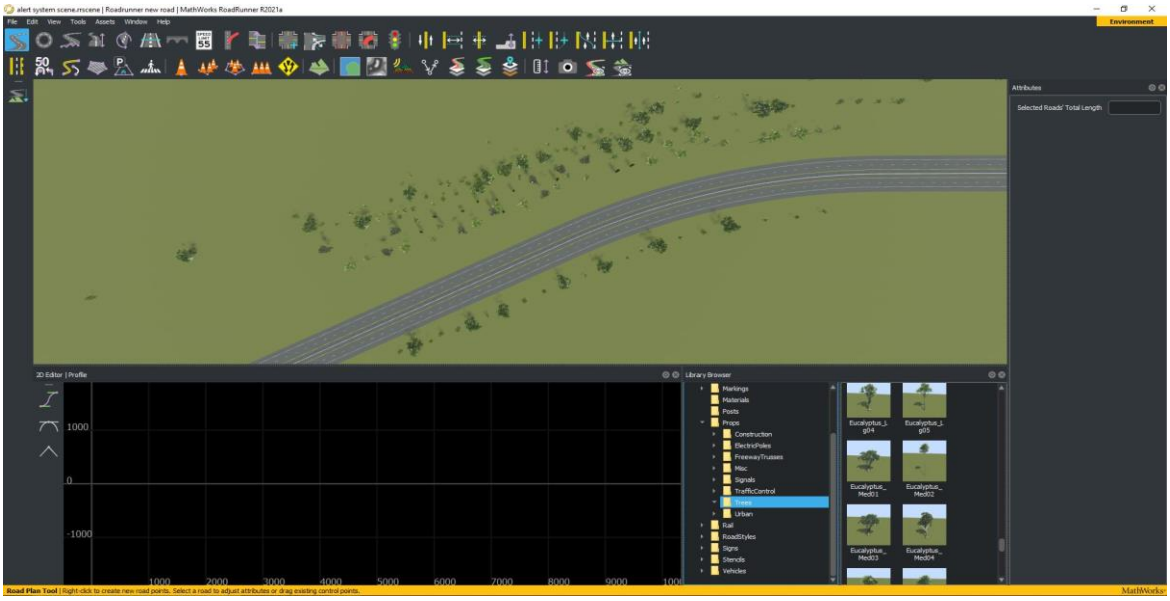

Figure 9. 45-degree curve with additional elements such as trees and vegetation

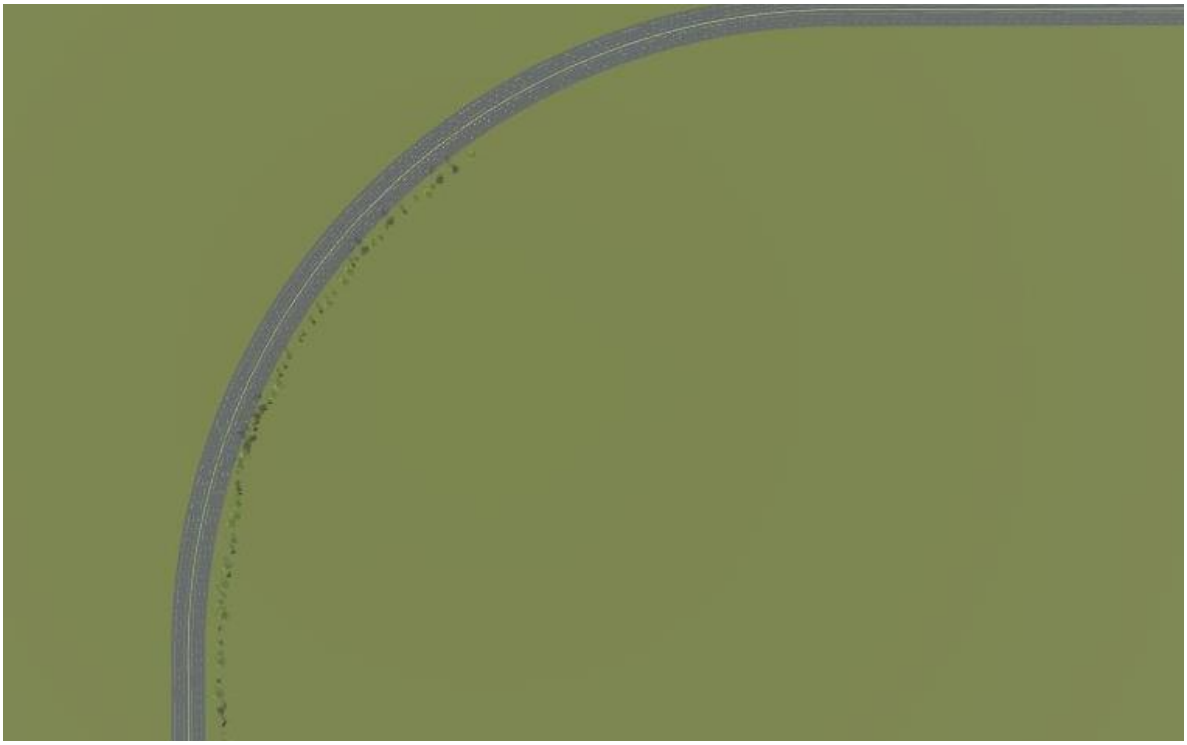

Figure 10. 90-degree curve

After importing the environment into the Unity, cars and waypoints have been added. Cars and waypoints were created using RealisticCarControllerV3. RealisticCarController (RCC) AI is a module that lets us create full functioning vehicles within the Unity. This module uses AI for running the vehicles, and the cars that are being created by this module, follow the waypoints from our inputs. For a smooth route, waypoints with the least possible gap between them has been created. Figure 11 shows a snapshot of the cars that have been imported. In figure 12, the right bottom corner shows the view from the camera, and the whole picture shows that the camera is installed on the back of a truck.

Figure 11. A picture of Unity showing the cars in simulator

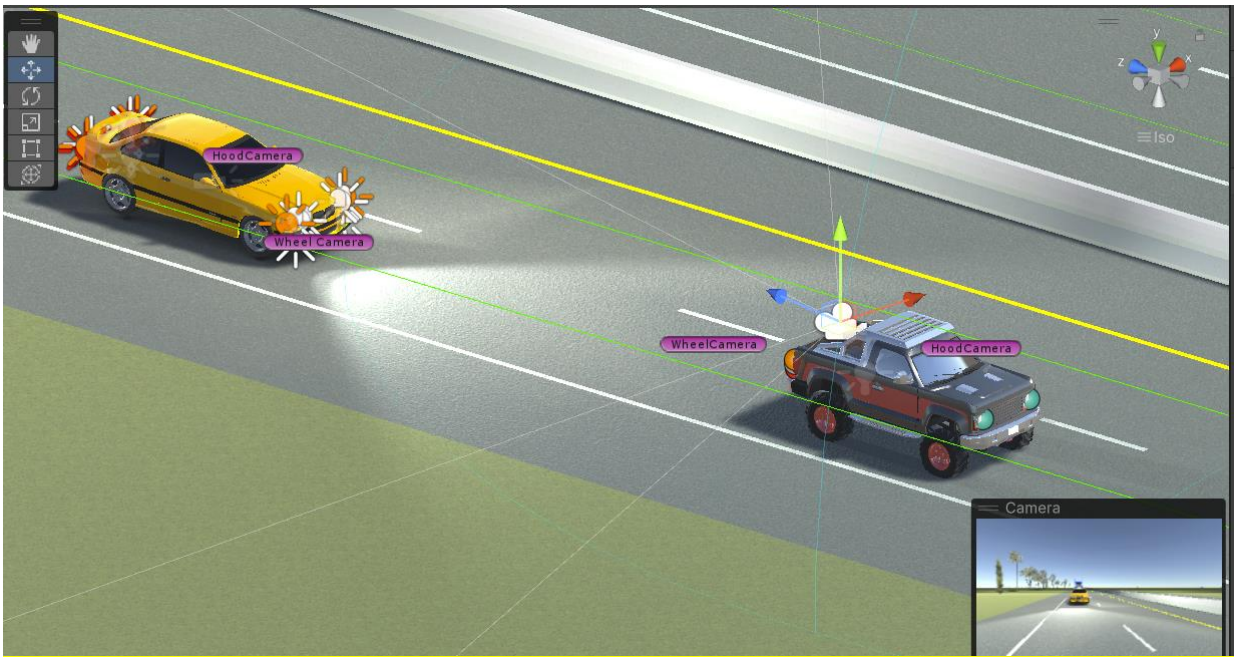

Figure 12. The right bottom corner shows the view from the camera, and the whole picture shows that the camera is installed on back of a truck.

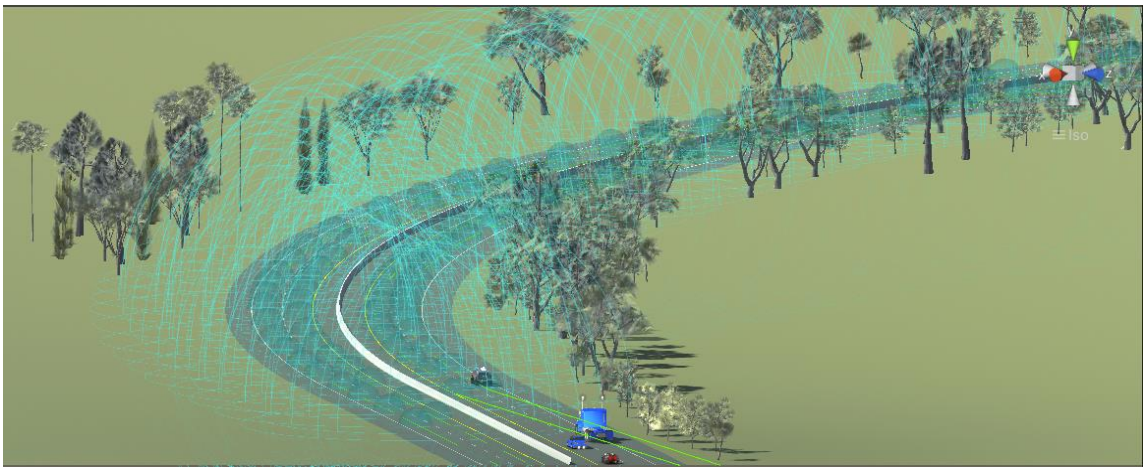

Figure 13. A picture of waypoints created in the Unity

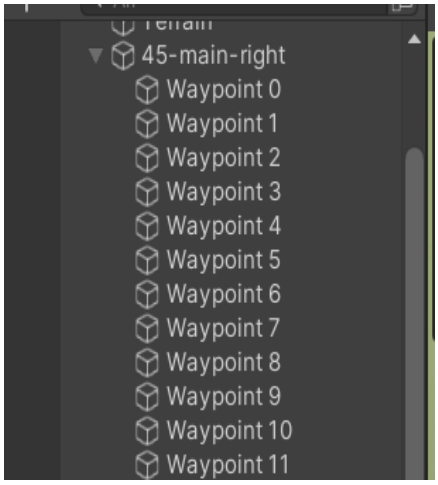

Figure 14. Unity menu containing waypoints created

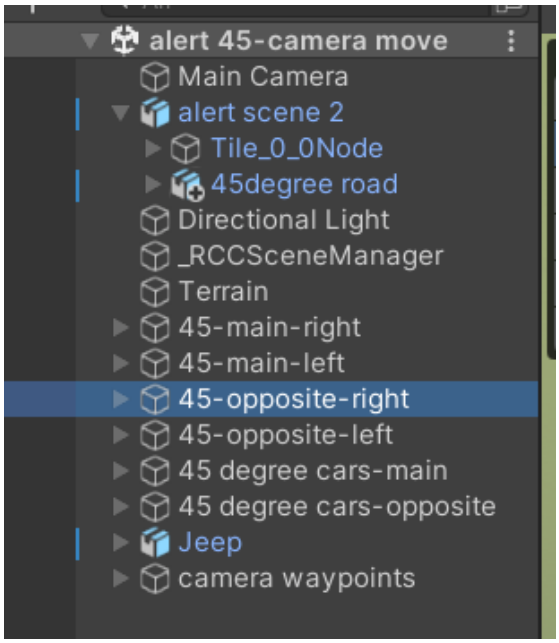

Figure 15. Unity menu with all the waypoints for different directions and lanes

For each of the curves (45 degrees and 90 degrees), 4 waypoint series (each one containing about 45 waypoints) for 4 lanes have been created. Figure 14 shows waypoints in one of the waypoint containers in the Unity and figure 15 shows waypoint containers. Two main lanes and two opposite lanes. After creating

the waypoints, they have been put in RCC AI Waypoints container function in RCC AI so that the cars can use these waypoints (figure 16).

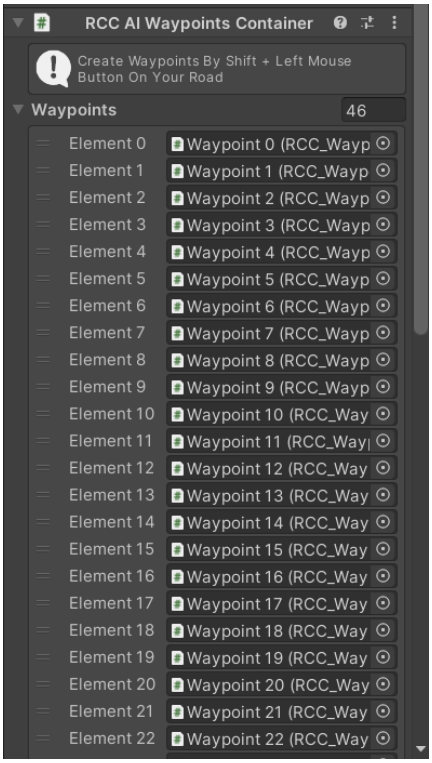

Figure 16. RCC AI waypoints

Two situations for each curve have been created. One situation with a camera mounted on an AI car, which other cars can see and avoid a crash. Another situation with the same concept has been created except that the car with a camera is a ghost car and the other cars can't see it and they can pass through. The difference between these two categories is traffic smoothness. The first category is a little bit more chaotic, because cars need to change lanes and they are usually at a high speed, and we wanted to see crashes happening. The second category is to see how this alarm system triggers in smooth traffic. In order to avoid bias,

we created random scenarios. Random cars have been placed randomly and are assigned a different speed each time for each simulation run. The number of vehicles were also random in each run. Overall, 30 scenarios have been created.

Realistic Car Controller has a variety of features for controlling vehicles. Some features and controls from this module have been calibrated, particularly the Ray Angle, Ray Distance, and Speed Limit. Ray Angle is basically the degree angles that a car can see, and Ray Distance is a distance a car can see to take maneuvers. Numerous runs and tests have been done to come up with different Ray Angles and Ray Distances for different cars because each type of car has different physics, dimensions, acceleration, power, reaction time etc.

Speed Limit is a speed that a car can't pass, but it can drive below that. Speed Limit has been changed for all cars in each scenario for variation and randomness. Some cars with speed limits of 90-100 have been included to assess aggressive driving. Overall, efforts for a harmonious flow speed have been done meaning that there are scenarios which cars speed limit are set to about 40, and there are also scenarios where all cars speed limit are set to 70 with some variations. Figure 17 shows a portion of the RCC AI controller setting in the Unity.

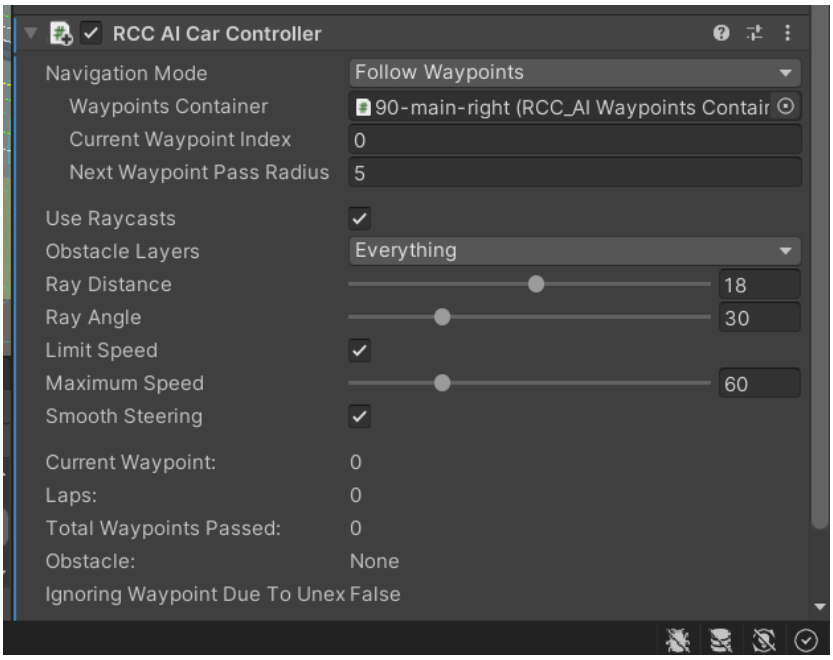

Figure 17. A portion of the RCC AI controller settings

3.2 Computer Vision Development

Before beginning the DeepStream methodology, it's better to define some terms which will be used a lot in this section.

Pipeline:

Data pipeline is a series of connected elements which the output of one element is the input of another one. These elements can be executed consecutively or in parallel (32).

GStreamer:

GStreamer is a multimedia framework that is based on pipelines and links a wide range of media processing systems in order solve and compute complex workflows. With GStreamer you can handle media components such as video and audio playback, streaming, recording, and editing. GStreamer has a plug-in architecture which can be implemented as shared libraries. GStreamer has functions for registering and loading plug-ins which provides the base classes or fundamentals of all classes. You can load plug-in libraries to support a wide range of containers, codecs and formats (33). Figure 18 shows an example of a GStreamer pipeline

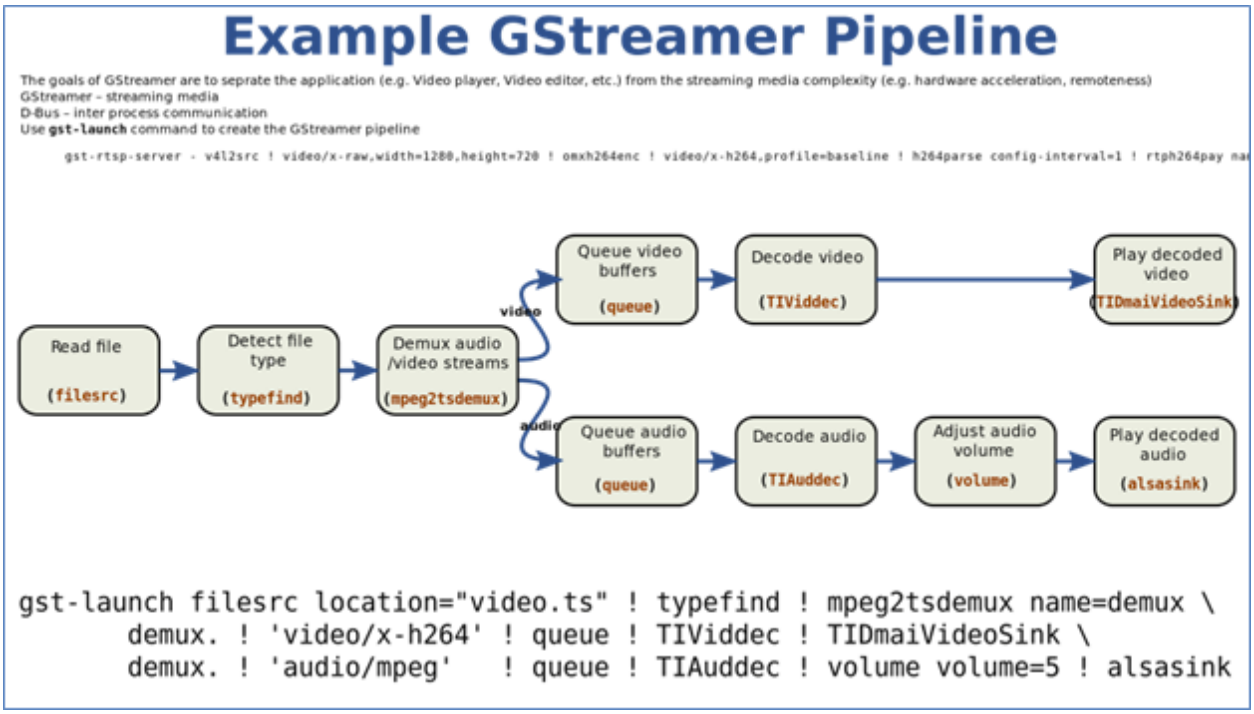

Figure 18. An example of a GStreamer Pipeline (33)

DeepStream:

Built on top of the GStreamer is the optimized architecture known as DeepStream. DeepStream uses a variety of accelerators while maintaining the zero-memory copy between plugins to deliver the best performance possible. DeepStream provides building pieces that can be used to build an efficient pipeline for video analytics in the form of GStreamer plugins. A comprehensive streaming analytics toolbox for AI-based multi-sensor processing, video, audio, and image processing is provided by NVIDIA's DeepStream SDK. For a range of object identification, picture classification, and instance segmentation-based AI models, DeepStream offers remarkable performance (34).

Gst-Nvdsanalytics:

This is one of many plugins that DeepStream has. This plugin is for doing analysis on metadata which include NVINFER (primary detector) and NVTRACKER. We can do a wide range of analytics and focus on Line Crossing. This plugin can handle multiple streams independently (35).

Bounding Boxes:

Bounding box is the rectangle boundaries of a detected object. Bounding boxes coordinates are defined as left, top, width, and height. Figure 19 shows the location of x and y coordinates. In some contexts, they use (x1, y1) for (left, top), and (x2, y2) for (left + width, top + height).

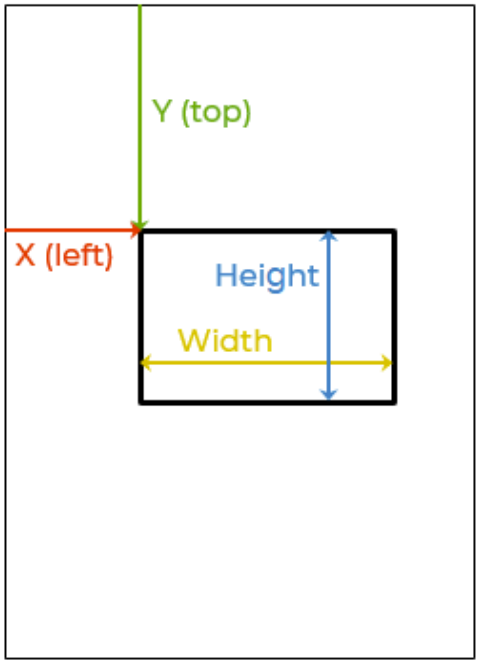

Figure 19. Bounding box demonstration

YOLO:

YOLO or You Only Look Once is a neural network for real-time object detection system which can be used for images or videos. It is an algorithm which includes localization of objects in the image or frame and predicting the object's class. YOLO uses only convolutional layers which makes it a fully convolutional network (FCN) (36).

Why use DeepStream?

The TMA alert system needs a GPU to run computer vision AI to detect cars and then trigger the alarm when needed. We chose Nvidia's Jetson Xavier AGX because of the processing power that it provides. It's a small device that can be

used inside a truck which is connected to a camera that is installed on the back of a TMA. The camera captures live video from the back of the TMA and since it is connected to the GPU, live analysis can be done on the video to trigger the alarm. Besides using DeepStream, other options for vehicle detection such as YOLO was available. I compared YOLO-V3-tiny to DeepStream and table 4 are the results for Jetson Xavier AGX (Note that both runs have been on the same 720p video for object detection):

Table 4. Speed comparison between YOLO-V3 tiny and DeepStream

Method	**FPS**
YOLO V3 tiny	20-22
DeepStream	29-30

DeepStream has a lot of advantages over YOLO or OpenCV. YOLO is at core only for image detector, so it's general use is object detection. OpenCV is just a library to work with images and uses machine learning to work with the images and videos. However, DeepStream can be utilized not only for object detection, but also you can do variety of tasks such as conversion of formats, working with multiple streams and so on with pipelines. Since the DeepStream outperformed and has much more advantages over other methods, it has been used for the rest of this study.

Defining the safety zone in Simulator:

As stated before, the goal is to develop an alarm system that notifies the drivers when they are close to a collision. For this purpose, on the recorded videos that was extracted from simulator runs, a safety zone has been defined meaning that if they are in the safety zone, they should be alarmed to change lanes or brake. Inside the DeepStream, some lines have been drawn to form a trapezoid. The targets are the vehicles on the same lane as the TMA truck, so when defining the safety zone, the other lane was ignored. As it is shown in the figure 20, we are only working with the same lane as the TMA truck. Since this is on the recorded simulator videos and it is not the real-time real-life scenario, and since we are working in the lab, instead of an alarm, changing the color of bounding boxes is our triggering action. When vehicles cross the defined boundary, their bounding box changes color from red to blue. To do this, NVDSANALYTICS plugin has been used inside DeepStream.

Figure 20. Safety Zone defined in the bottom half of the screen

For the outputs, Frame Number, Object ID and Class ID were extracted from the detected objects in the safety zone and were printed in a log file. Frame Number is basically the Nth frame in the video, meaning that if a video is 60 seconds and the video is 30fps, we will have 1800 frames. So overall, it means that in the Nth frame, we extracted the object ID and Class ID of the detected objects in the safety zone. Object ID is a unique number which is assigned every time DeepStream detects a new object. Class ID is the class of the detected object. In this tracker, the class ID can be 0 to 3 which means car, bicycle, person, and road sign respectively.

Safety Zone on real TMA video:

The same approach has been used on real TMA field video as well. After trying different coordinates to see what area is good for the safety zone that does not trigger the other cars in the adjacent lane, the coordinates of the safety zone have been modified to match our TMA truck position in the field video. Figure 21 shows the safety zone in the TMA video:

Figure 21. Safety Zone is defined on the left bottom corner of the real-world TMA videos

After defining the safety zone, we need to tell the DeepStream that data needs to be extracted here. In order to do that, we need some conditions and then printing to a file. Our conditions here are the boundaries of the safety zone. If a car

crosses these boundaries, it gets triggered. Since our defined zone is a trapezoid, we need to work with line formula. As we know the formula of a line is this:

$$y - y1 = \left(\frac{y1-y2}{x1-x2}\right)(x - x1) \quad (37)$$

So, our condition will look like this:

$$x > (y - y1)\left(\frac{\Delta x}{\Delta y}\right) + x1$$

These conditions for diagonal lines plus conditions for “top” position (y) will be used for four points of a bounding box. The y should be in a range like: y1<y<y2. Y1 is the top pixel coordinate and y2 is the bottom pixel coordinate. The logic is that even if one point is in the zone, it gets triggered. The coordinates of a bounding box are defined as figure 22:

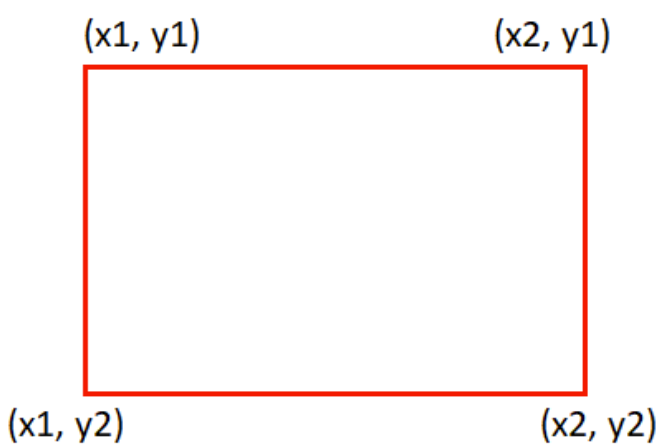

Figure 22. Coordinates for the bounding boxes. If any of these points cross the safety zone, the bounding box color changes.

CHAPTER 4 TEST AND RESULTS

After running the computer vision program on 30 simulation runs, the total number of unique object IDs in the whole simulation run inside of the safety zone has been extracted and then has been analyzed with Python programming. Furthermore, the results have been reviewed manually and printed on tables 5 to 9.

Table 5. Results from 90-degree curve simulator recorded videos

Simulation number	Number of triggered vehicles	True positive	False alarms	Accuracy
1	14	10	4	71.4%
2	12	9	3	75.0%
3	19	14	5	73.7%
4	16	12	4	75.0%
5	17	13	4	76.5%
6	21	15	6	71.4%
7	20	15	5	75.0%

Table 6. Results from 90-degree curve mounted on a ghost truck simulator recorded videos

Simulation number	Number of triggered vehicles	True positive	False alarms	Accuracy
1	23	18	5	78.3%
2	19	16	3	84.2%
3	19	16	3	84.2%
4	17	14	3	82.4%
5	23	18	5	78.3%
6	21	16	5	76.2%
7	26	20	6	76.9%
8	22	16	6	72.7%

Table 7. Results from 45-degree curve simulator recorded videos

Simulation number	Number of triggered vehicles	True positive	False alarms	Accuracy
1	20	15	5	75.0%
2	20	15	5	75.0%
3	31	22	9	71.0%

4	35	22	13	62.9%
5	25	18	7	72.0%
6	12	9	3	75.0%
7	11	8	3	72.7%
8	14	10	4	71.4%

Table 8. Results from 45-degree curve mounted on a ghost truck simulator recorded videos

Simulation number	Number of triggered vehicles	True positive	False alarms	Accuracy
1	21	17	4	81.0%
2	19	16	3	84.2%
3	30	24	6	80.0%
4	25	21	4	84.0%
5	29	23	6	79.3%
6	28	22	6	78.6%
7	32	26	6	81.3%

Table 9. Average accuracies

	45-degree	45-degree on a ghost car	90-degree	90-degree on a ghost car	Total
Average accuracy	71.9%	81.2%	74%	79.1%	76.6%

The results show that the accuracy is substantially higher (about 8% more accurate on average) when the camera was mounted on a ghost truck. The reason is that in the other two scenario, we had numerous crashes with the TMA truck, so that the TMA truck was rotated a bit at the time of hit, thereby it was detecting more cars by accident from the other lane which caused more false alarms than when the camera was mounted on a ghost truck that doesn't have any interaction with the traffic. Figure 23 shows an example of a false alarm in our simulator. In the real world, there are not as many crashes happening, and that means that we expect the results of the real world be close to what we got from when the camera was mounted on a ghost truck (with about 80% accuracy).

Figure 23. Example of a false alarm triggering a car in a different lane

After watching all the videos to check the results, it was found that there were times that the bounding box of a detected car had disappeared (which is a common issue in object detection methods), but when a car enters the safety zone, at least there was a single frame that our system detected the car in the safety zone and changed its bounding box color. This issue occurs when the object gets too close to the screen. So, overall, this proves that our program is capable of triggering the alarm system to notify the 100% of the cars that enter the safety zone. However, false alarm is a serious issue when the safety zone is not properly adjusted. Figure 24 to 27 are snapshots from our simulator recorded videos.

Figure 24. Example of the safety zone triggering bounding box colors

Figure 25. Example of triggering bounding box colors when they enter the safety zone

Figure 26. Example of the same car when they get too close to the camera and not get detected it anymore.

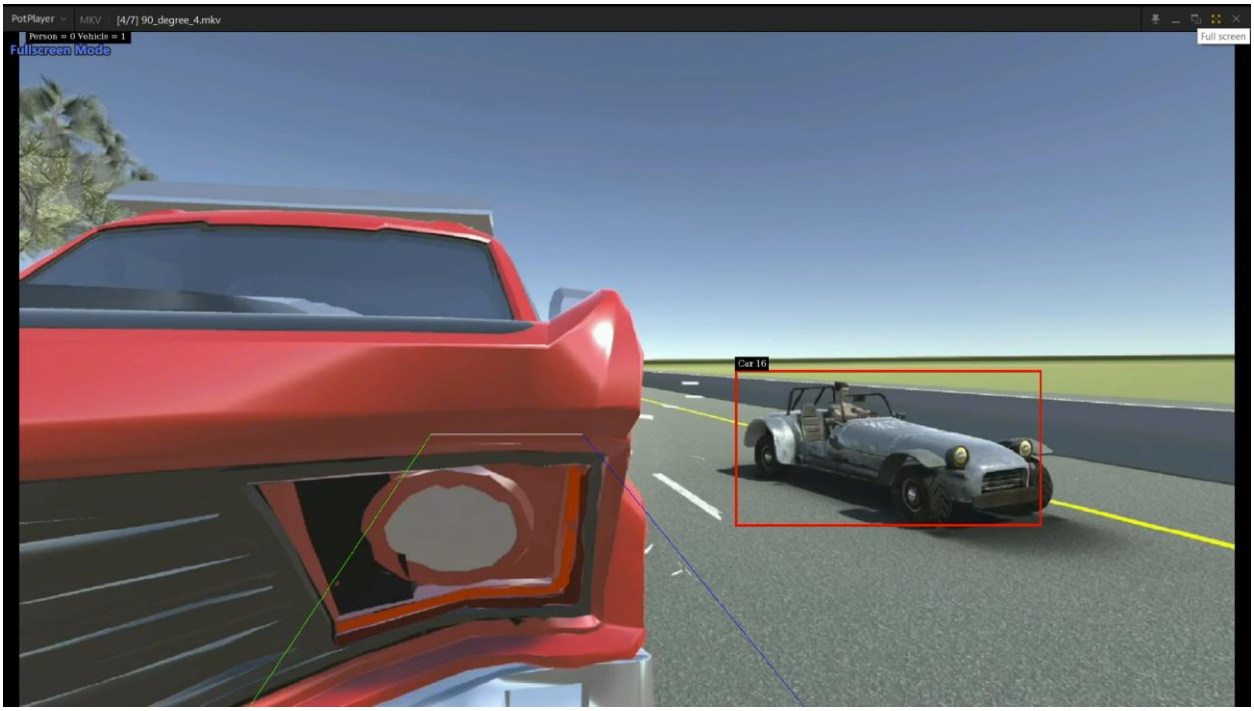

Figure 27. Example of high speed cars crashing with the TMA

From our simulator videos, figure 24 and 25 shows examples of detecting vehicles and changing their bounding box colors (triggering) when they enter the safety zone. Figure 26 is an example of when cars get too close to the camera and our system could not detect the vehicle (note that the car got triggered the first frame that entered the safety zone). Figure 27 shows an example of an aggressive driver with high speed that crashes into the TMA truck.

So far, it was proved that our system is a good system to alarm the drivers in simulation. Now we can go forward and test it on field videos from TMAs. One of the main challenges we faced in the simulator videos is not having a proper safety zone or let’s say we should have had a dynamic safety zone. When the safety zone is fixed, if the truck rotates to some degree, even the vehicles on the other lane can trigger the alarm which causes false alarms. The figure 28 below shows how the defined safety zone in the field test video looks like.

Figure 28. Defined safety zone in real-world TMA videos

The TMA results look so promising. We had a 16-minute video from a TMA. Although the video was so shaky, it detected all the cars that entered the safety zone. As expected, there were some conditions that we got false alarms. Below are examples of a false alarm (figure 29) and a true positive alarm (figure 30) in the field video respectively. Then table 10 shows the results from analyzing our field video.

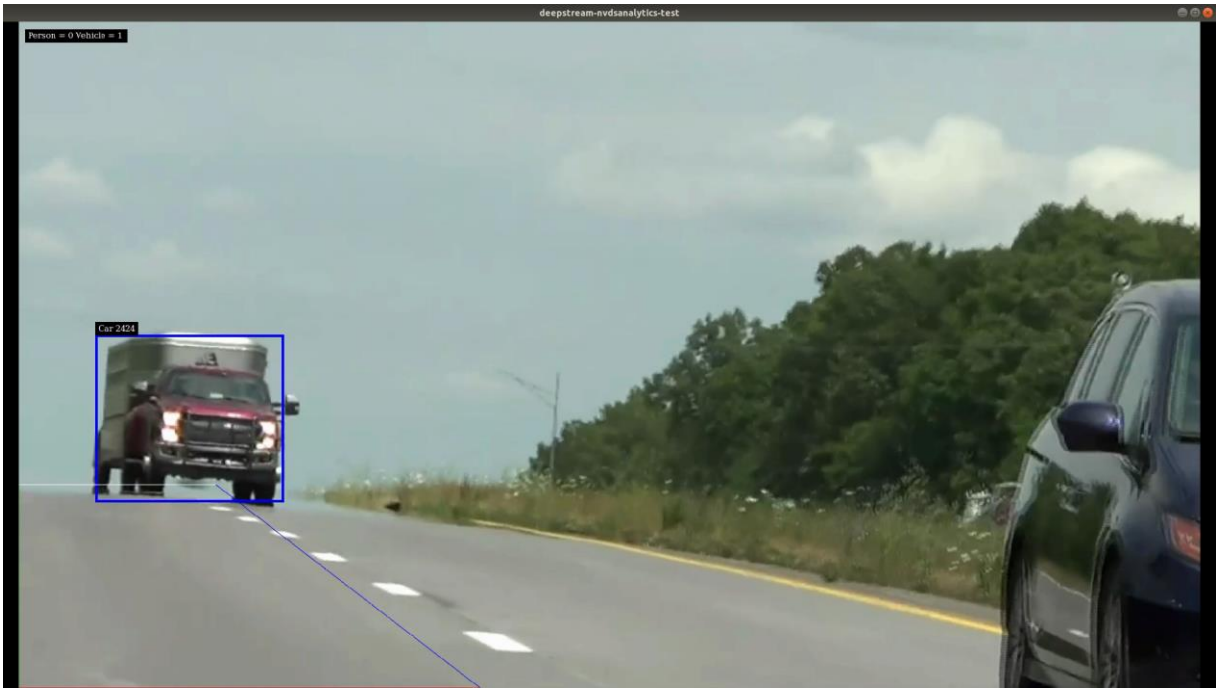

Figure 29. False alarm example on the field test video. The truck was passing from the other lane

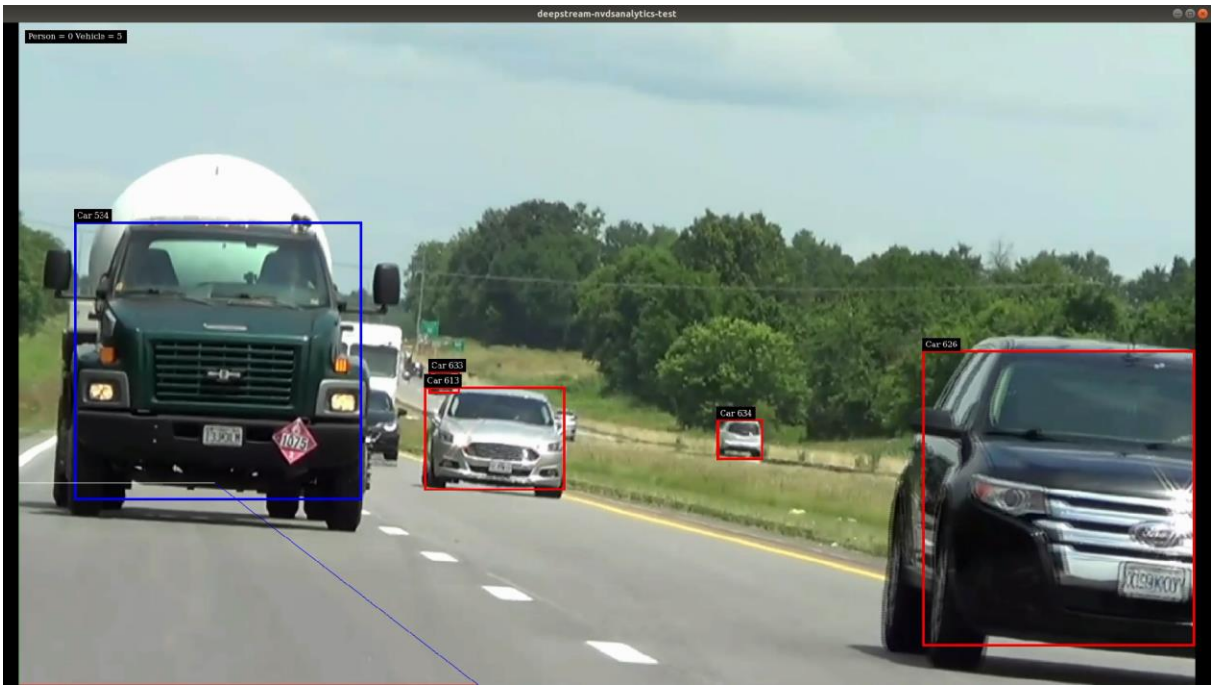

Figure 30. Example of a true positive alarm when it entered the safety zone on the same lane as the TMA truck in the field test video

Table 10. Results from TMA field video

	Number of triggered vehicles	True positive	False alarms	Accuracy
TMA field video	20	7	13	65%

Although all the cars that entered the safety zone got triggered, results from table 10 show that we had 7 out of 20 (35%) false alarms which is quite high. This is again due to having a fixed safety zone. As it is shown in figure 29, when the geometry of the road changes, like curves or hills, the safety zone needs to be changed relatively. The rest of the vehicles that got triggered, they were changing lanes, or they changed lanes after entering the safety zone. Figure 30 shows an example of a true positive alarm.

From simulator and field results we can conclude that the accuracy of our developed system relates to how the safety zone is defined, because under different conditions, different results can be captured. In the first 14 minutes of the field video, there was no false alarms, and all the false alarms happened right after the TMA truck started to come down from a hill in a span of 2 minutes. It can be said that our developed system worked 100% accurate for the first 14 minutes of a 16-minute video. And that's because we defined the safety zone based on "before the hill" section and then after that the geometry of the road changed, false alarms started to occur. In addition to that, there were some situations where it could not detect the vehicles because it was being shaky, but overall, it detected

the cars that entered the safety zone in this particular video. Figure 31 shows when DeepStream is unable to detect vehicles when the camera shakes a lot.

Figure 31. Not detecting cars due to the shakes of the camera

CHAPTER 5 CONCLUSION AND FUTURE WORKS

The original contribution of this thesis focuses on development and test of an alarm system in work zones using simulator and computer vision to improve safety and efficiency. The combination of simulator and field studies is important. Simulator offers a safe environment to test new designs or technologies that sometimes is expensive or rare in the real world. Field study on the other hand, has the advantage of providing realistic data to be used for validation. Our developed method contained simulating close-to-life scenarios with 45 degrees and 90 degrees curves with various speeds and traffic volumes to include randomness, designing scenarios to include crashes, and at last, developing a system that can alarm the drivers who enter a safety zone that is defined by code. By testing our system on simulator videos and real TMA videos, we found that our developed system is promising and would be able to alarm the drivers who enter the safety zone. In the future, this developed system needs to be tested in real time for a long period of time to see how much this system is effective for crashes in real time compared to before using this system.

The camera that was used for field study, was a regular camera. In order to have the best results, it is better to use cameras that have top notch stabilizers, shockproof and waterproof to overcome harsh situations or unexpected weather conditions. The field videos were shaky and that puts the TMA truck in danger because it might not detect the cars if the road pavement is not smooth, or the

weather is windy which adds additional shakes to the overall TMA truck shakes. Therefore, professional action/sport cameras such as GoPro or Insta360 which have the best stabilizers or even additional stabilizers like gimbals is needed to overcome this issue.

Since this approach is new, we can expand this method and use new technologies such as depth camera instead of a regular camera to get additional data such as vehicle speed and alarm those who are going at high speeds before they reach the safety zone, so they would have enough time to react. Similarly for vehicles which are already reducing their speed and enter slightly in the safety zone, there would be no need for an alarm. Depth camera can be useful to tackle these issues.

In the future, this system can be implemented on real-time streams from back of a TMA. As stated before, one of the main challenges is the lack of a dynamic safety zone. And this is not a simple issue when the road marks and paintings are not sometimes readable. So, if we conquer this challenge, our automated system will be a good system to work with in the work zones and hopefully a lot of lives will be saved.

REFERENCE

1. Ranney TA, Garrott WR, Goodman MJ. NHTSA DRIVER DISTRACTION RESEARCH: PAST, PRESENT, AND FUTURE.

2. Driver_Fatigue_and_Road_Accidents.

3. Automated Flagger Assistance Device. [cited 2022 Jul 26]; Available from: https://www.autoflagger.com/

4. Be wary of snow plows during wintry weather. [cited 2022 Jul 26]; Available from: https://www.galioninquirer.com/news/37380/wary-snow-plows-wintry-weather

5. Truck-mounted attenuators are designed to save lives. [cited 2022 Jul 26]; Available from: https://spasafetysystems.com/truck-mounted-attenuators/

6. Mukhtar A, Xia L, Tang TB. Vehicle Detection Techniques for Collision Avoidance Systems: A Review. Vol. 16, IEEE Transactions on Intelligent Transportation Systems. Institute of Electrical and Electronics Engineers Inc.; 2015. p. 2318–38.

7. Xiao L, Gao F. A comprehensive review of the development of adaptive cruise control systems. Vehicle System Dynamics. 2010 Oct;48(10):1167–92.

8. van Arem B, van Driel CJG, Visser R. The impact of cooperative adaptive cruise control on traffic-flow characteristics. IEEE Transactions on Intelligent Transportation Systems. 2006 Dec;7(4):429–36.

9. Milanes V, Shladover SE, Spring J, Nowakowski C, Kawazoe H, Nakamura M. Cooperative adaptive cruise control in real traffic situations. IEEE Transactions on Intelligent Transportation Systems. 2014 Feb;15(1):296–305.

10. Broggi A, Cerri P, Ghidoni S, Grisleri P, Jung HG. A new approach to urban pedestrian detection for automatic braking. IEEE Transactions on Intelligent Transportation Systems. 2009 Dec;10(4):594–605.

11. Chen BC, Luan BC, Lee K. Design of lane keeping system using adaptive model predictive control. In: IEEE International Conference on Automation Science and Engineering. IEEE Computer Society; 2014. p. 922–6.

12. Son YS, Kim W, Lee SH, Chung CC. Robust multirate control scheme with predictive virtual lanes for lane-keeping system of autonomous highway driving. IEEE Transactions on Vehicular Technology. 2015 Aug 1;64(8):3378–91.

13. Levinson J, Askeland J, Becker J, Dolson J, Held D, Kammel S, et al. Towards fully autonomous driving: Systems and algorithms. In: IEEE Intelligent Vehicles Symposium, Proceedings. 2011. p. 163–8.

14. Zou Q, Cao Y, Li Q, Mao Q, Wang S. CrackTree: Automatic crack detection from pavement images. Pattern Recognition Letters. 2012 Feb 1;33(3):227–38.

15. Zhang S, Sun C, Advisor D. Investigation of Smart Work Zone Technologies Using Mixed Simulator and Field Studies. 2020.

16. of Transportation Federal Highway Administration USD. FHWA Work Zone Facts and Statistics [Internet]. Available from: https://ops.fhwa.dot.gov/wz/resources/facts_stats.htm

17. Sun C, Feng H. Safety Analysis of Truck Mounted Attenuators in Mobile Work Zones Using Differential Analysis.

18. of Transportation MD. TMA Crashes and Associated Employee Injuries -1h [Internet]. Available from: https://www.modot.org/tma-crashes-and-associated-employee-injuries-1h

19. Ullman GL, Iragavarapu V. Analysis of expected crash reduction benefits and costs of truck-mounted attenuator use in work zones. Vol. 2458, Transportation Research Record. National Research Council; 2014. p. 74–7.

20. Cottrell BH. Investigation of Truck Mounted Attenuator (TMA) Crashes in Work Zones in Virginia [Internet]. Available from: http://www.virginiadot.org/vtrc/main/online_reports/pdf/16-r7.pdf

21. Brown H, Sun C, Cope T. Evaluation of mobile work zone alarm systems. Transportation Research Record. 2015;2485:42–50.

22. Pourfalatoun S, Miller EE. User perceptions of automated Truck-Mounted attenuators: Implications on work zone safety. Traffic Injury Prevention. 2021;22(5):413–8.

23. Yanpeng Cao ; A. Renfrew ; P. Cook. Vehicle motion analysis based on a monocular vision system.

24. Cheon M, Lee W, Yoon C, Park M. Vision-Based Vehicle Detection System With Consideration of the Detecting Location. IEEE Transactions on Intelligent Transportation Systems. 2012 Apr 11;13(3):1243–52.

25. Lan J, Zhang M. A new vehicle detection algorithm for real-time image processing system. In: ICCASM 2010 - 2010 International Conference on Computer Application and System Modeling, Proceedings. 2010.

26. Jin LS, Gu BY, Wang R ben, Guo L, Zhao YB, Li LH. Preceding vehicle detection based on multi-characteristics fusion. In: 2006 IEEE International Conference on Vehicular Electronics and Safety, ICVES. 2006. p. 356–60.

27. Intensity and Edge-Based Symmetry Detection with an Application to Car-Following - ScienceDirect [Internet]. [cited 2022 Jul 19]. Available from: https://www.sciencedirect.com/science/article/abs/pii/S1049966083710375

28. Tsai LW, Hsieh JW, Fan KC. Vehicle detection using normalized color and edge map. IEEE Transactions on Image Processing. 2007 Mar;16(3):850–64.

29. Chen YL, Chen YH, Chen CJ, Wu BF. Nighttime vehicle detection for driver assistance and autonomous vehicles. In: Proceedings - International Conference on Pattern Recognition. 2006. p. 687–90.

30. Chang P, Hirvonen D, Camus T, Southall B. Stereo-based object detection, classification, and quantitative evaluation with automotive applications. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. IEEE Computer Society; 2005.

31. Betke M, Haritaoglu E, Davis LS. Machine Vision and Applications Real-time multiple vehicle detection and tracking from a moving vehicle. Machine Vision and Applications. 2000.

32. Pipeline (computing). In: Wikipedia [Internet]. [cited 2022 Jul 17]. Available from: https://en.wikipedia.org/wiki/Pipeline_(computing)

33. GStreamer. In: Wikipedia [Internet]. [cited 2022 Jul 17]. Available from: https://en.wikipedia.org/wiki/GStreamer

34. Deepstream SDK [Internet]. [cited 2022 Jul 17]. Available from: https://developer.nvidia.com/deepstream-sdk

35. Gst-nvdsanalytics. In [cited 2022 Jul 27]. Available from: https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_plugin_gst-nvdsanalytics.html

36. YOLO: Real-Time Object Detection. [cited 2022 Jul 20]; Available from: https://pjreddie.com/darknet/yolo/

37. Line (geometry). In [cited 2022 Jul 26]. Available from: https://en.wikipedia.org/wiki/Line_(geometry)

Appendix

Code used in this thesis:

```
#include <gst/gst.h>
#include <glib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <sys/time.h>
#include <iostream>
#include <vector>
#include <unordered_map>
#include <sstream>
#include <cuda_runtime_api.h>
#include "gstnvdsmeta.h"
#include "nvds_analytics_meta.h"
#ifndef PLATFORM_TEGRA
#include "gst-nvmessage.h"
#endif

#define MAX_DISPLAY_LEN 64

#define PGIE_CLASS_ID_VEHICLE 0
#define PGIE_CLASS_ID_PERSON 2

/* The muxer output resolution must be set if the input streams will be of
 * different resolution. The muxer will scale all the input frames to this
 * resolution. */
#define MUXER_OUTPUT_WIDTH 1920
#define MUXER_OUTPUT_HEIGHT 1080

/* Muxer batch formation timeout, for e.g. 40 millisec. Should ideally be set
```

```
 * based on the fastest source's framerate. */
#define MUXER_BATCH_TIMEOUT_USEC 40000

#define TILED_OUTPUT_WIDTH 1920
#define TILED_OUTPUT_HEIGHT 1080

/* NVIDIA Decoder source pad memory feature. This feature signifies that source
 * pads having this capability will push GstBuffers containing cuda buffers. */
#define GST_CAPS_FEATURES_NVMM "memory:NVMM"

gchar pgie_classes_str[4][32] = { "Vehicle", "TwoWheeler", "Person",
  "RoadSign"
};

struct TriggerInfo
{
    int trigger_frame;
    int trigger_stream;
    uint32_t trigger_value;
    std::string trigger_class;
};



/* nvdsanalytics_src_pad_buffer_probe  will extract metadata received on tiler
sink pad
 * and extract nvanalytics metadata etc. */
static GstPadProbeReturn
nvdsanalytics_src_pad_buffer_probe (GstPad * pad, GstPadProbeInfo * info,
    gpointer u_data)
{
    GstBuffer *buf = (GstBuffer *) info->data;
    guint num_rects = 0;
    NvDsObjectMeta *obj_meta = NULL;
    guint vehicle_count = 0;
    guint person_count = 0;
```

```
    NvDsMetaList * l_frame = NULL;
    NvDsMetaList * l_obj = NULL;
    NvDsDisplayMeta *display_meta = NULL;

    NvDsBatchMeta *batch_meta = gst_buffer_get_nvds_batch_meta (buf);

    struct TriggerInfo zoneInfo[1000];
    int zcnt = 0;
    struct TriggerInfo lineInfo[1000];
    int lcnt = 0;

    for (l_frame = batch_meta->frame_meta_list; l_frame != NULL;l_frame = l_frame-
>next)
    {
        NvDsFrameMeta *frame_meta = (NvDsFrameMeta *) (l_frame->data);
        std::stringstream out_string;
        vehicle_count = 0;
        num_rects = 0;
        person_count = 0;
        int offset = 0;

        int myframenumber = frame_meta->frame_num;
        int mystreamnumber = frame_meta->pad_index;




        for (l_obj = frame_meta->obj_meta_list; l_obj != NULL;l_obj = l_obj->next)
        {
            obj_meta = (NvDsObjectMeta *) (l_obj->data);
            if (obj_meta->class_id == PGIE_CLASS_ID_VEHICLE) {
                vehicle_count++;
                num_rects++;
            }
            if (obj_meta->class_id == PGIE_CLASS_ID_PERSON) {
                person_count++;
                num_rects++;
```

```
            }



            FILE *abc;
            char *afilename = "logs/obj_class_results.logs";
            abc = fopen(afilename, "a");


                int bbox_bottom_left = obj_meta->rect_params.top + obj_meta-
>rect_params.height;
           int bbox_bottom_right = bbox_bottom_left + obj_meta->rect_params.width;



             //if (bbox_bottom_left > 650 && obj_meta->rect_params.left > 300
&& obj_meta->rect_params.top < 900 )
            //if (bbox_bottom_left > 650 && bbox_bottom_right < 800 )
            //if (bbox_bottom_left > 650 && obj_meta->rect_params.left > 650 &&
bbox_bottom_right < 890 )
           //simulation//if ((bbox_bottom_left > 650 && obj_meta->rect_params.left
> 650 && obj_meta->rect_params.left < 900) || (bbox_bottom_left > 650 &&
bbox_bottom_right > 650 && bbox_bottom_right < 900 ) || (obj_meta->rect_params.top
> 650 && obj_meta->rect_params.left > 650 && obj_meta->rect_params.left < 900) ||
(obj_meta->rect_params.top > 650 && bbox_bottom_right > 650 && bbox_bottom_right
< 900))
            ///TMA///////////////
           if ((obj_meta->rect_params.left < ((43/32)*(obj_meta->rect_params.top-
750)+320) && obj_meta->rect_params.top > 750 && obj_meta->rect_params.top < 1080)
||
               (obj_meta->rect_params.left < ((43/32)*(bbox_bottom_left-750)+320)
&& bbox_bottom_left > 750 && bbox_bottom_left < 1080 ) ||
               (bbox_bottom_right < ((43/32)*(obj_meta->rect_params.top-750)+320)
&& obj_meta->rect_params.top > 750 && obj_meta->rect_params.top < 1080) ||
                   (bbox_bottom_right < ((43/32)*(bbox_bottom_left-750)+320) &&
bbox_bottom_left > 750 && bbox_bottom_left < 1080))


            {
              obj_meta->rect_params.has_bg_color = 0;
              obj_meta->rect_params.border_color.red = 0.0;
              obj_meta->rect_params.border_color.green = 0.0;
              obj_meta->rect_params.border_color.blue = 1.0;
```

```
              obj_meta->rect_params.border_color.alpha = 1.0;


             fprintf(abc, "%d,%d,%d\n",frame_meta->frame_num, obj_meta->object_id,
obj_meta->class_id );
              //fprintf(abc, "%d,%d,%d\n",1,2,3);
            }


            fclose(abc);



            // Access attached user meta for each object
                for  (NvDsMetaList  *l_user_meta  =  obj_meta->obj_user_meta_list;
l_user_meta != NULL;
                    l_user_meta = l_user_meta->next) {
                NvDsUserMeta *user_meta = (NvDsUserMeta *) (l_user_meta->data);
                                                if(user_meta->base_meta.meta_type    ==
NVDS_USER_OBJ_META_NVDSANALYTICS)
                {
                    NvDsAnalyticsObjInfo * user_meta_data = (NvDsAnalyticsObjInfo
*)user_meta->user_meta_data;
                    if (user_meta_data->dirStatus.length()){
                        g_print ("object %lu moving in %s\n", obj_meta->object_id,
user_meta_data->dirStatus.c_str());
                    }
                }
            }
        }


        display_meta = nvds_acquire_display_meta_from_pool(batch_meta);
        // /* Line Testing for simulation*/
        // display_meta->num_lines = 4; ////change to 0 to disable it


        // NvOSD_LineParams *line_params  = &display_meta->line_params[0];


        // line_params->x1 = 650;
        // line_params->y1 = 650;
        // line_params->x2 = 890;
```

```
        // line_params->y2 = 650;

        // line_params->line_width = 1;
        // line_params->line_color.red = 1.0;
        // line_params->line_color.green = 1.0;
        // line_params->line_color.blue = 1.0;
        // line_params->line_color.alpha = 1.0;

        // NvOSD_LineParams *line_params1  = &display_meta->line_params[1];
        // line_params1->x1 = 1500;
        // line_params1->y1 = 1400;
        // line_params1->x2 = 150;
        // line_params1->y2 = 1400;

        // line_params1->line_width = 1;
        // line_params1->line_color.red = 1.0;
        // line_params1->line_color.green = 0.0;
        // line_params1->line_color.blue = 0.0;
        // line_params1->line_color.alpha = 1.0;

        // NvOSD_LineParams *line_params2  = &display_meta->line_params[2];
        // line_params2->x1 = 150;
        // line_params2->y1 = 1400;
        // line_params2->x2 = 650;
        // line_params2->y2 = 650;

        // line_params2->line_width = 1;
        // line_params2->line_color.red = 0.0;
        // line_params2->line_color.green = 1.0;
        // line_params2->line_color.blue = 0.0;
        // line_params2->line_color.alpha = 1.0;


        // NvOSD_LineParams *line_params3  = &display_meta->line_params[3];
        // line_params3->x1 = 1500;
```

```
        // line_params3->y1 = 1400;
        // line_params3->x2 = 890;
        // line_params3->y2 = 650;

        // line_params3->line_width = 1;
        // line_params3->line_color.red = 0.0;
        // line_params3->line_color.green = 0.0;
        // line_params3->line_color.blue = 1.0;
        // line_params3->line_color.alpha = 1.0;

        // nvds_add_display_meta_to_frame(frame_meta, display_meta);
        ////////////////////////////////////////////////////////////
                                ////////////////////line       testing       for
TMA//////////////////////////////////////////
        display_meta->num_lines = 4; ////change to 0 to disable it

        NvOSD_LineParams *line_params  = &display_meta->line_params[0];

        line_params->x1 = 0;
        line_params->y1 = 750;
        line_params->x2 = 320;
        line_params->y2 = 750;

        line_params->line_width = 1;
        line_params->line_color.red = 1.0;
        line_params->line_color.green = 1.0;
        line_params->line_color.blue = 1.0;
        line_params->line_color.alpha = 1.0;

        NvOSD_LineParams *line_params1  = &display_meta->line_params[1];
        line_params1->x1 = 750;
        line_params1->y1 = 1080;
        line_params1->x2 = 0;
        line_params1->y2 = 1080;
```

```
        line_params1->line_width = 1;
        line_params1->line_color.red = 1.0;
        line_params1->line_color.green = 0.0;
        line_params1->line_color.blue = 0.0;
        line_params1->line_color.alpha = 1.0;

        NvOSD_LineParams *line_params2  = &display_meta->line_params[2];
        line_params2->x1 = 0;
        line_params2->y1 = 1080;
        line_params2->x2 = 0;
        line_params2->y2 = 750;

        line_params2->line_width = 1;
        line_params2->line_color.red = 0.0;
        line_params2->line_color.green = 1.0;
        line_params2->line_color.blue = 0.0;
        line_params2->line_color.alpha = 1.0;


        NvOSD_LineParams *line_params3  = &display_meta->line_params[3];
        line_params3->x1 = 750;
        line_params3->y1 = 1080;
        line_params3->x2 = 320;
        line_params3->y2 = 750;

        line_params3->line_width = 1;
        line_params3->line_color.red = 0.0;
        line_params3->line_color.green = 0.0;
        line_params3->line_color.blue = 1.0;
        line_params3->line_color.alpha = 1.0;

        nvds_add_display_meta_to_frame(frame_meta, display_meta);

        ////////////////////////////////////////////////////////////////////////////////
//////
```

```
        /* Iterate user metadata in frames to search analytics metadata */
         for (NvDsMetaList * l_user = frame_meta->frame_user_meta_list;l_user !=
NULL; l_user = l_user->next)
        {
            NvDsUserMeta *user_meta = (NvDsUserMeta *) l_user->data;
                                    if     (user_meta->base_meta.meta_type     !=
NVDS_USER_FRAME_META_NVDSANALYTICS)
                continue;

            /* convert to  metadata */
            NvDsAnalyticsFrameMeta *meta =
                (NvDsAnalyticsFrameMeta *) user_meta->user_meta_data;
            /* Get the labels from nvdsanalytics config file */
            for (std::pair<std::string, uint32_t> status : meta->objInROIcnt){
                out_string << "Objs in ROI ";
                out_string << status.first;
                out_string << " = ";
                out_string << status.second;

                //if (status.second > 0)
                {
                  zoneInfo[zcnt].trigger_stream = mystreamnumber;
                  zoneInfo[zcnt].trigger_frame = myframenumber;
                  zoneInfo[zcnt].trigger_class = status.first;
                  zoneInfo[zcnt].trigger_value = status.second;

                  zcnt++;
                }
            }
            for (std::pair<std::string, uint32_t> status : meta->objLCCumCnt){
                out_string << " LineCrossing Cumulative ";
                out_string << status.first;
                out_string << " = ";
                out_string << status.second;
            }
```

```
            for (std::pair<std::string, uint32_t> status : meta->objLCCurrCnt){
                out_string << " LineCrossing Current Frame ";
                out_string << status.first;
                out_string << " = ";
                out_string << status.second;

                //if (status.second > 0)
                {
                  lineInfo[lcnt].trigger_stream = mystreamnumber;
                  lineInfo[lcnt].trigger_frame = myframenumber;
                  lineInfo[lcnt].trigger_class = status.first;
                  lineInfo[lcnt].trigger_value = status.second;
                  lcnt++;
                }
            }
            for (std::pair<std::string, bool> status : meta->ocStatus){
                out_string << " Overcrowding status ";
                out_string << status.first;
                out_string << " = ";
                out_string << status.second;
            }
        }

                                ///////////////////////BEGIN       DRAWING       ON
SCREEN//////////////////////////////////////////
        display_meta = nvds_acquire_display_meta_from_pool(batch_meta);
        NvOSD_TextParams *txt_params  = &display_meta->text_params[0];
        display_meta->num_labels = 1;
        txt_params->display_text = (char*)g_malloc0 (MAX_DISPLAY_LEN);
        offset = snprintf(txt_params->display_text, MAX_DISPLAY_LEN, "Person = %d
", person_count);
         offset = snprintf(txt_params->display_text + offset , MAX_DISPLAY_LEN,
"Vehicle = %d ", vehicle_count);

        /* Now set the offsets where the string should appear */
        txt_params->x_offset = 10;
```

```
        txt_params->y_offset = 12;

        /* Font , font-color and font-size */
        txt_params->font_params.font_name = "Serif";
        txt_params->font_params.font_size = 10;
        txt_params->font_params.font_color.red = 1.0;
        txt_params->font_params.font_color.green = 1.0;
        txt_params->font_params.font_color.blue = 1.0;
        txt_params->font_params.font_color.alpha = 1.0;

        /* Text background color */
        txt_params->set_bg_clr = 1;
        txt_params->text_bg_clr.red = 0.0;
        txt_params->text_bg_clr.green = 0.0;
        txt_params->text_bg_clr.blue = 0.0;
        txt_params->text_bg_clr.alpha = 1.0;

        nvds_add_display_meta_to_frame(frame_meta, display_meta);




                                ///////////////////////END       DRAWING       ON
SCREEN///////////////////////////////////////




        g_print ("Frame Number = %d of Stream = %d, Number of objects = %d "
                "Vehicle Count = %d Person Count = %d %s\n",
            frame_meta->frame_num, frame_meta->pad_index,
            num_rects, vehicle_count, person_count, out_string.str().c_str());



        //// Trigger Alert
        FILE *afp, *aafp;
```

```
        char *afilename = "action/alert.action";
        char *aafilename = "action/line.alert";
        afp = fopen(afilename, "w+");

        //Trigger Line-Crossings Alert
        fprintf(afp, "-1"); //Turn everything off
        fclose(afp);

        //FILE *afp;
                                                //char          *afilename          =
"/home/mymark/Desktop/TMA_AlertMonitor/action/alert.action";

        // afp = fopen(afilename, "w+");
        // aafp = fopen(aafilename, "a");

         int alertZones[5] = {0,0,0,0,0}; //Trigger Zones

        //Filter initial trigger logic
        for(int i = 0; i < lcnt; i++)
        {
          afp = fopen(afilename, "w+");
          std::string mZones = lineInfo[i].trigger_class;
          if(mZones == "Zone1")
          {
            alertZones[0] = 9;
            fprintf(afp, "%d\n", 9 );
          }
          else if (mZones == "Zone2")
          {
            alertZones[1] = 7;
            fprintf(afp, "%d\n", 4);
          }
          else if (mZones == "Zone3")
          {
            alertZones[2] = 5;
            fprintf(afp, "%d\n", 3 );
```

```
        }
        else if (mZones == "Zone4")
        {
          alertZones[3] = 3;
          fprintf(afp, "%d\n", 1 );
        }
        else
        {
          alertZones[4] = 1;
          fprintf(afp, "%d\n", 0 );
        }

        fclose(afp);

      }

      // for(int i = 0; i < 5; i++)
      // {
      //   if(alertZones[i] != 1)
      //   {
      //     int v = alertZones[i];
      //     fprintf(afp, "%d\n", v );
      //     fprintf(aafp, "%d\n", v );
      //     break;
      //   }
      // }

      // fclose(afp);
      // fclose(aafp);

      //// Logs
      FILE *lfp;
      char *lfilename = "logs/line_results.logs";
      lfp = fopen(lfilename, "a");
```

```
        FILE *zfp;
        char *zfilename = "logs/zone_results.logs";
        zfp = fopen(zfilename, "a");

        for(int i = 0; i < lcnt; i++)
        {
                fprintf(lfp,  "%d,%d,%s,%d\n",frame_meta->frame_num,  frame_meta-
>pad_index, lineInfo[i].trigger_class.c_str(), lineInfo[i].trigger_value);
        }

        for(int i = 0; i < zcnt; i++)
        {
                fprintf(zfp,  "%d,%d,%s,%d\n",frame_meta->frame_num,  frame_meta-
>pad_index,zoneInfo[i].trigger_class.c_str(), zoneInfo[i].trigger_value);
        }

        fclose(lfp);
        fclose(zfp);

    }
    return GST_PAD_PROBE_OK;
}

static gboolean
bus_call (GstBus * bus, GstMessage * msg, gpointer data)
{
  GMainLoop *loop = (GMainLoop *) data;
  switch (GST_MESSAGE_TYPE (msg)) {
    case GST_MESSAGE_EOS:
      g_print ("End of stream\n");
      g_main_loop_quit (loop);
      break;
    case GST_MESSAGE_WARNING:
    {
      gchar *debug;
      GError *error;
```

```
      gst_message_parse_warning (msg, &error, &debug);
      g_printerr ("WARNING from element %s: %s\n",
          GST_OBJECT_NAME (msg->src), error->message);
      g_free (debug);
      g_printerr ("Warning: %s\n", error->message);
      g_error_free (error);
      break;
    }
    case GST_MESSAGE_ERROR:
    {
      gchar *debug;
      GError *error;
      gst_message_parse_error (msg, &error, &debug);
      g_printerr ("ERROR from element %s: %s\n",
          GST_OBJECT_NAME (msg->src), error->message);
      if (debug)
        g_printerr ("Error details: %s\n", debug);
      g_free (debug);
      g_error_free (error);
      g_main_loop_quit (loop);
      break;
    }
#ifndef PLATFORM_TEGRA
    case GST_MESSAGE_ELEMENT:
    {
      if (gst_nvmessage_is_stream_eos (msg)) {
        guint stream_id;
        if (gst_nvmessage_parse_stream_eos (msg, &stream_id)) {
          g_print ("Got EOS from stream %d\n", stream_id);
        }
      }
      break;
    }
#endif
    default:
```

```
      break;
  }
  return TRUE;
}

static void
cb_newpad (GstElement * decodebin, GstPad * decoder_src_pad, gpointer data)
{
  g_print ("In cb_newpad\n");
  GstCaps *caps = gst_pad_get_current_caps (decoder_src_pad);
  const GstStructure *str = gst_caps_get_structure (caps, 0);
  const gchar *name = gst_structure_get_name (str);
  GstElement *source_bin = (GstElement *) data;
  GstCapsFeatures *features = gst_caps_get_features (caps, 0);

  /* Need to check if the pad created by the decodebin is for video and not
   * audio. */
  if (!strncmp (name, "video", 5)) {
    /* Link the decodebin pad only if decodebin has picked nvidia
     * decoder plugin nvdec_*. We do this by checking if the pad caps contain
     * NVMM memory features. */
    if (gst_caps_features_contains (features, GST_CAPS_FEATURES_NVMM)) {
      /* Get the source bin ghost pad */
      GstPad *bin_ghost_pad = gst_element_get_static_pad (source_bin, "src");
      if (!gst_ghost_pad_set_target (GST_GHOST_PAD (bin_ghost_pad),
              decoder_src_pad)) {
        g_printerr ("Failed to link decoder src pad to source bin ghost pad\n");
      }
      gst_object_unref (bin_ghost_pad);
    } else {
      g_printerr ("Error: Decodebin did not pick nvidia decoder plugin.\n");
    }
  }
}
```

```
static void
decodebin_child_added (GstChildProxy * child_proxy, GObject * object,
    gchar * name, gpointer user_data)
{
  g_print ("Decodebin child added: %s\n", name);
  if (g_strrstr (name, "decodebin") == name) {
    g_signal_connect (G_OBJECT (object), "child-added",
        G_CALLBACK (decodebin_child_added), user_data);
  }
}

static GstElement *
create_source_bin (guint index, gchar * uri)
{
  GstElement *bin = NULL, *uri_decode_bin = NULL;
  gchar bin_name[16] = { };

  g_snprintf (bin_name, 15, "source-bin-%02d", index);
  /* Create a source GstBin to abstract this bin's content from the rest of the
   * pipeline */
  bin = gst_bin_new (bin_name);

  /* Source element for reading from the uri.
   * We will use decodebin and let it figure out the container format of the
   * stream and the codec and plug the appropriate demux and decode plugins. */
  uri_decode_bin = gst_element_factory_make ("uridecodebin", "uri-decode-bin");

  if (!bin || !uri_decode_bin) {
    g_printerr ("One element in source bin could not be created.\n");
    return NULL;
  }

  /* We set the input uri to the source element */
  g_object_set (G_OBJECT (uri_decode_bin), "uri", uri, NULL);
```

```
  /* Connect to the "pad-added" signal of the decodebin which generates a
   * callback once a new pad for raw data has beed created by the decodebin */
  g_signal_connect (G_OBJECT (uri_decode_bin), "pad-added",
      G_CALLBACK (cb_newpad), bin);
  g_signal_connect (G_OBJECT (uri_decode_bin), "child-added",
      G_CALLBACK (decodebin_child_added), bin);

  gst_bin_add (GST_BIN (bin), uri_decode_bin);

  /* We need to create a ghost pad for the source bin which will act as a proxy
   * for the video decoder src pad. The ghost pad will not have a target right
   * now. Once the decode bin creates the video decoder and generates the
   * cb_newpad callback, we will set the ghost pad target to the video decoder
   * src pad. */
  if (!gst_element_add_pad (bin, gst_ghost_pad_new_no_target ("src",
              GST_PAD_SRC))) {
    g_printerr ("Failed to add ghost pad in source bin\n");
    return NULL;
  }

  return bin;
}

int
main (int argc, char *argv[])
{
  GMainLoop *loop = NULL;
  GstElement *pipeline = NULL, *streammux = NULL, *sink = NULL, *pgie = NULL,
              *nvtracker = NULL, *nvdsanalytics = NULL,
      *nvvidconv = NULL, *nvosd = NULL, *tiler = NULL,
      *queue1, *queue2, *queue3, *queue4, *queue5, *queue6, *queue7;
  GstElement *transform = NULL;
  GstBus *bus = NULL;
  guint bus_watch_id;
  GstPad *nvdsanalytics_src_pad = NULL;
```

```
guint i, num_sources;
guint tiler_rows, tiler_columns;
guint pgie_batch_size;

FILE *abc;
char *afilename = "logs/obj_class_results.logs";
abc = fopen(afilename, "w+");
fclose(abc);

FILE *zfp;
char *zfilename = "logs/zone_results.logs";
zfp = fopen(zfilename, "w+");
fclose(zfp);


int current_device = -1;
cudaGetDevice(&current_device);
struct cudaDeviceProp prop;
cudaGetDeviceProperties(&prop, current_device);

/* Check input arguments */
if (argc < 2) {
  g_printerr ("Usage: %s <uri1> [uri2] ... [uriN] \n", argv[0]);
  return -1;
}
num_sources = argc - 1;

/* Standard GStreamer initialization */
gst_init (&argc, &argv);
loop = g_main_loop_new (NULL, FALSE);

/* Create gstreamer elements */
/* Create Pipeline element that will form a connection of other elements */
pipeline = gst_pipeline_new ("nvdsanalytics-test-pipeline");
```

```
/* Create nvstreammux instance to form batches from one or more sources. */
streammux = gst_element_factory_make ("nvstreammux", "stream-muxer");

if (!pipeline || !streammux) {
  g_printerr ("One element could not be created. Exiting.\n");
  return -1;
}
gst_bin_add (GST_BIN (pipeline), streammux);

for (i = 0; i < num_sources; i++) {
  GstPad *sinkpad, *srcpad;
  gchar pad_name[16] = { };
  GstElement *source_bin = create_source_bin (i, argv[i + 1]);

  if (!source_bin) {
    g_printerr ("Failed to create source bin. Exiting.\n");
    return -1;
  }

  gst_bin_add (GST_BIN (pipeline), source_bin);

  g_snprintf (pad_name, 15, "sink_%u", i);
  sinkpad = gst_element_get_request_pad (streammux, pad_name);
  if (!sinkpad) {
    g_printerr ("Streammux request sink pad failed. Exiting.\n");
    return -1;
  }

  srcpad = gst_element_get_static_pad (source_bin, "src");
  if (!srcpad) {
    g_printerr ("Failed to get src pad of source bin. Exiting.\n");
    return -1;
  }

  if (gst_pad_link (srcpad, sinkpad) != GST_PAD_LINK_OK) {
```

```
    g_printerr ("Failed to link source bin to stream muxer. Exiting.\n");
    return -1;
  }

  gst_object_unref (srcpad);
  gst_object_unref (sinkpad);
}

/* Use nvinfer to infer on batched frame. */
pgie = gst_element_factory_make ("nvinfer", "primary-nvinference-engine");

/* Use nvtracker to track detections on batched frame. */
nvtracker = gst_element_factory_make ("nvtracker", "nvtracker");

/* Use nvdsanalytics to perform analytics on object */
nvdsanalytics = gst_element_factory_make ("nvdsanalytics", "nvdsanalytics");

/* Use nvtiler to composite the batched frames into a 2D tiled array based
 * on the source of the frames. */
tiler = gst_element_factory_make ("nvmultistreamtiler", "nvtiler");

/* Use convertor to convert from NV12 to RGBA as required by nvosd */
nvvidconv = gst_element_factory_make ("nvvideoconvert", "nvvideo-converter");

/* Create OSD to draw on the converted RGBA buffer */
nvosd = gst_element_factory_make ("nvdsosd", "nv-onscreendisplay");

/* Add queue elements between every two elements */
queue1 = gst_element_factory_make ("queue", "queue1");
queue2 = gst_element_factory_make ("queue", "queue2");
queue3 = gst_element_factory_make ("queue", "queue3");
queue4 = gst_element_factory_make ("queue", "queue4");
queue5 = gst_element_factory_make ("queue", "queue5");
queue6 = gst_element_factory_make ("queue", "queue6");
queue7 = gst_element_factory_make ("queue", "queue7");
```

```
  /* Finally render the osd output */
  if(prop.integrated) {
    transform = gst_element_factory_make ("nvegltransform", "nvegl-transform");
  }
  sink = gst_element_factory_make ("nveglglessink", "nvvideo-renderer");

  if (!pgie || !nvtracker || !nvdsanalytics || !tiler || !nvvidconv ||
      !nvosd || !sink || !queue1 || !queue2 || !queue3 || !queue4 || !queue5 ||
      !queue6 || !queue7
      ) {
    g_printerr ("One element could not be created. Exiting.\n");
    return -1;
  }

  if(!transform && prop.integrated) {
    g_printerr ("One tegra element could not be created. Exiting.\n");
    return -1;
  }

  g_object_set (G_OBJECT (streammux), "width", MUXER_OUTPUT_WIDTH, "height",
      MUXER_OUTPUT_HEIGHT, "batch-size", num_sources,
      "batched-push-timeout", MUXER_BATCH_TIMEOUT_USEC, NULL);

  /* Configure the nvinfer element using the nvinfer config file. */
  g_object_set (G_OBJECT (pgie),
      "config-file-path", "nvdsanalytics_pgie_config.txt", NULL);

  /* Configure the nvtracker element for using the particular tracker algorithm.
*/
  g_object_set (G_OBJECT (nvtracker),
                        "ll-lib-file",       "/opt/nvidia/deepstream/deepstream-
6.0/lib/libnvds_nvmultiobjecttracker.so",
                   "ll-config-file",     "../../../../samples/configs/deepstream-
app/config_tracker_NvDCF_perf.yml",
      "tracker-width", 640, "tracker-height", 480,
```

```
      NULL);

  /* Configure the nvdsanalytics element for using the particular analytics config
file*/
  g_object_set (G_OBJECT (nvdsanalytics),
      "config-file", "config_nvdsanalytics.txt",
       NULL);

  /* Override the batch-size set in the config file with the number of sources. */
  g_object_get (G_OBJECT (pgie), "batch-size", &pgie_batch_size, NULL);
  if (pgie_batch_size != num_sources) {
    g_printerr
        ("WARNING: Overriding infer-config batch-size (%d) with number of sources
(%d)\n",
        pgie_batch_size, num_sources);
    g_object_set (G_OBJECT (pgie), "batch-size", num_sources, NULL);
  }

  tiler_rows = (guint) sqrt (num_sources);
  tiler_columns = (guint) ceil (1.0 * num_sources / tiler_rows);
  /* we set the tiler properties here */
  g_object_set (G_OBJECT (tiler), "rows", tiler_rows, "columns", tiler_columns,
      "width", TILED_OUTPUT_WIDTH, "height", TILED_OUTPUT_HEIGHT, NULL);

  g_object_set (G_OBJECT (sink), "qos", 0, NULL);

  /* we add a message handler */
  bus = gst_pipeline_get_bus (GST_PIPELINE (pipeline));
  bus_watch_id = gst_bus_add_watch (bus, bus_call, loop);
  gst_object_unref (bus);

  /* Set up the pipeline */
  /* we add all elements into the pipeline */
  if(prop.integrated) {
    gst_bin_add_many (GST_BIN (pipeline), queue1,  pgie, queue2, nvtracker, queue3,
            nvdsanalytics , queue4, tiler, queue5,
```

```
            nvvidconv, queue6, nvosd, queue7, transform, sink,
        NULL);

    /* we link the elements together, with queues in between
    * nvstreammux -> nvinfer -> nvtracker -> nvdsanalytics -> nvtiler ->
    * nvvideoconvert -> nvosd -> transform -> sink
    */
    if (!gst_element_link_many (streammux,queue1, pgie , queue2, nvtracker,
          queue3, nvdsanalytics, queue4, tiler, queue5,
          nvvidconv, queue6, nvosd, queue7, transform, sink, NULL)) {
      g_printerr ("Elements could not be linked. Exiting.\n");
      return -1;
    }
  }
  else {
    gst_bin_add_many (GST_BIN (pipeline), queue1, pgie, queue2,
                  nvtracker, queue3, nvdsanalytics, queue4, tiler, queue5,
                  nvvidconv, queue6, nvosd, queue7, sink, NULL);
    /* we link the elements together
    * nvstreammux -> nvinfer -> nvtracker -> nvdsanalytics -> nvtiler ->
    * nvvideoconvert -> nvosd -> sink
    */
    if (!gst_element_link_many (streammux, queue1, pgie, queue2, nvtracker,
        queue3, nvdsanalytics, queue4, tiler, queue5, nvvidconv, queue6,
        nvosd, queue7, sink, NULL)) {
      g_printerr ("Elements could not be linked. Exiting.\n");
      return -1;
    }
  }

  /* Lets add probe to get informed of the meta data generated, we add probe to
   * the sink pad of the nvdsanalytics element, since by that time, the buffer
   * would have had got all the metadata.
   */
  nvdsanalytics_src_pad = gst_element_get_static_pad (nvdsanalytics, "src");
```

```
  if (!nvdsanalytics_src_pad)
    g_print ("Unable to get src pad\n");
  else
    gst_pad_add_probe (nvdsanalytics_src_pad, GST_PAD_PROBE_TYPE_BUFFER,
        nvdsanalytics_src_pad_buffer_probe, NULL, NULL);
  gst_object_unref (nvdsanalytics_src_pad);

  /* Set the pipeline to "playing" state */
  g_print ("Now playing:");
  for (i = 0; i < num_sources; i++) {
    g_print (" %s,", argv[i + 1]);
  }
  g_print ("\n");
  gst_element_set_state (pipeline, GST_STATE_PLAYING);

  /* Wait till pipeline encounters an error or EOS */
  g_print ("Running...\n");
  g_main_loop_run (loop);

  /* Out of the main loop, clean up nicely */
  g_print ("Returned, stopping playback\n");
  gst_element_set_state (pipeline, GST_STATE_NULL);
  g_print ("Deleting pipeline\n");
  gst_object_unref (GST_OBJECT (pipeline));
  g_source_remove (bus_watch_id);
  g_main_loop_unref (loop);
  return 0;
}
```