AUTOMATED TEST CASE GENERATION FOR SELF-DRIVING CARS USING CCTV VIDEOS

by
VATAN AKSOY TEZER

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfilment of the requirements for the degree of
Master of Science

Sabancı University
July 2021

AUTOMATED TEST CASE GENERATION FOR SELF-DRIVING CARS USING CCTV VIDEOS

Approved by:

Date of Approval: July 6, 2021

ABSTRACT

AUTOMATED TEST CASE GENERATION FOR SELF-DRIVING CARS USING CCTV VIDEOS

VATAN AKSOY TEZER

Computer Science and Engineering, M.S. THESIS, JULY 2021

Thesis Supervisor: Assoc. Prof. Dr. Cemal Yılmaz

Keywords: self-driving cars, simulator, image processing, artificial intelligence, automated test case generation

Self-driving cars are more and more included in our daily lives with recent advancements in the fields of artificial intelligence and robotics. Self-driving cars are typically tested in simulation and real-life with various types of tests in different scenarios. To ensure the safety of self-driving cars we must be able to conduct realistic test cases in simulation and real life. The most realistic and highly convincing tests include test cases from real-life accidents. However, currently, these tests are generated manually with humans still in the loop. We introduce a generic and scalable way to realistically generate automated test cases from any car accident video that is available. To show the flexibility of our study we use various YouTube videos that we have no prior information about for evaluation. The proposed method consists of three steps, namely analysis, scene reconstruction, and test case generation. Our method is fully automated and an end-to-end solution to generate test cases. In the analysis step, we run the input videos through an image processing pipeline that consists of six internal steps. Then we reconstruct the crash in a 3D physics engine. Finally, we generate various test cases from a set of automatically pre-defined or user-defined parameters. The test cases can be used with any autonomous driving stack that satisfies the communication requirements through a simulation bridge. We evaluate our results with a user study and a set of case study experiments that are conducted on the popular open-source autonomous driving stack, Apollo.

ÖZET

OTONOM ARAÇLAR İÇİN CCTV VIDEOLARI KULLANARAK OTOMATİK TEST SENARYOLARI OLUŞTURULMASI

VATAN AKSOY TEZER

Bilgisayar Bilimi ve Mühendisliği, YÜKSEK LİSANS TEZİ, Temmuz 2021

Tez Danışmanı: Doç. Dr. Cemal Yılmaz

Anahtar Kelimeler: otonom araçlar, simülatör, görüntü işleme, yapay zeka, otomatik test senaryoları oluşturma

Otonom araçlar, yapay zeka ve robotik alanlarındaki son gelişmelerle günlük hayatımıza giderek daha fazla dahil olmakta. Otonom araçlar tipik olarak simülasyonda ve gerçek hayatta farklı senaryolarda çeşitli testlerle test edilir. Otonom araçların güvenliğini sağlamak için simülasyonda ve gerçek hayatta gerçekçi test senaryoları yürütebilmeliyiz. Bu kapsamda en gerçekçi ve ikna edici testler, gerçek hayattaki kazalardan test senaryolarını içerir. Ancak şu anda bu testler, halen insanlar yardımı ile oluşturuluyor. Bu çalışmada mevcut herhangi bir araba kazası videosundan gerçekçi bir şekilde otomatik test senaryoları oluşturabilen genel ve ölçeklenebilir bir yöntem sunuyoruz. Çalışmamızın esnekliğini göstermek için, değerlendirme aşamasında hakkında önceden bilgi sahibi olmadığımız çeşitli YouTube videolarını kullanıyoruz. Önerilen yöntem, analiz, yeniden sahneyi canlandırma ve test senaryosu oluşturma olmak üzere üç adımdan oluşmaktadır. Geliştirilen yöntem tamamen otomatizedir ve test senaryoları oluşturmak için uçtan uca bir çözüm sunar. Analiz adımında, giriş videolarını altı farklı adımdan oluşan bir görüntü işleme yapısı üzerinden çalıştırıyoruz. Sonrasında kazayı üç boyutlu bir fizik simülasyonunda yeniden oluşturuyoruz. Son olarak, bir dizi otomatik olarak önceden tanımlanmış veya kullanıcı tanımlı parametrelerden çeşitli test senaryoları oluşturuyoruz. Oluşturulan test senaryoları, bir simülasyon köprüsü aracılığıyla iletişim gereksinimlerini karşılayan herhangi bir otonom sürüş sistemi ile kullanılabilir. Sonuçlarımızı bir kullanıcı çalışması ve popüler açık kaynaklı otonom sürüş sistemi olan Apollo üzerinde yürütülen bir dizi test ile değerlendiriyoruz.

ACKNOWLEDGEMENTS

I would like to send my deepest thanks to my advisor, Assoc. Prof. Dr. Cemal Yılmaz, who was understanding, thoughtful and supportive through all my research including the hard times of COVID-19. Without the positivity, encouragement and countless advices from him I wouldn't be able to complete my thesis.

I owe much of the completion of this thesis to my family, who kept me motivated towards my goal while loving unconditionally and respecting the roadmap I drew for myself, supporting in every they can.

I would like to thank all my friends that helped me through tough times, with technical and non technical talks, showing their support for my success. My homemate Yusuf Demiroğlu for making countless talks to help me get through when I was down, and Abdul Rahman Dabbour and Sencer Yazıcı for their great friendship and mentorship through technical problems, deserves as a special recognition.

All the user study participants, that voluntarily spend precious times of theirs to help me complete my user studies.

Finally to Sabancı University and all my lecturers that fulfilled my dreams to become a Computer Science and Engineering graduate, by tirelessly providing education. Sabancı made me a grew up in a professional, careful and knowledgeable way as a person.

To my family
Aileme

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

1. INTRODUCTION

1.1 Motivation

Developments in autonomous vehicle technology have the potential to change the way humanity lives in the future immensely. The technology on autonomous vehicles is growing and much more likely to continue growing over the near future. Numerous tech giants have seen the opportunity in autonomous vehicles and joined the competition including Google parent Alphabet's Waymo, Ford Motor Company's Argo AI, Elon Musk's Tesla, and China's Baidu. These companies have already been testing their autonomous vehicles on the roads, driving millions of miles.

Autonomous vehicles need to face a great number of strict tests to ensure the safety regulations are met, as similar with the other technologies that involve usage and interaction with humans. There are two main phases of autonomous vehicle testing: simulation and road tests. Hardware is always expensive, but when the hardware to be tested is an autonomous vehicle with precise sensors, it is even more expensive. In early phases of development, there usually exists a great number of edge cases and the software can make fatal mistakes. To avoid financial losses and endangering the test conductors, the tests are done in simulation environments, where there are no safety issues at all and the software failures don't cost lives or hardware losses. After the software is developed to be mature enough in simulation, real life tests start with the exact hardware and safety precautions.

There are already sophisticated test methods and environments that are being used by autonomous vehicle companies and researchers. But the test cases are usually hardcoded into the system, or randomly generated from user or NHTSA (National Highway Traffic Safety Administration) defined scenarios (Thorn, Kimmel & Chaka, 2018). Although one can cover as many scenarios in traffic as possible, it's hard

to evaluate how likely they are going to happen and if they cover the necessary safety requirements to declare an autonomous vehicle safe enough. Autonomous vehicles are already mature enough to perform quite successfully in most every day scenarios in a robust fashion, but the edge cases are usually the setbacks in autonomous vehicle software. These edge cases can include misleading signs, human driver faults, unobserved dynamic objects, and estimating the next events that are likely to occur. Whether it is an autonomous vehicle or a human driver, these edge cases can be hard to deal with and unfortunately can result in accidents. Therefore, our idea is to use real life accidents for test case generation and came up with the method of potentially converting every single car accident video available in the world to a test case that can support different autonomous driving stacks, which would mean a substantially large pool of test cases.

The idea of using accidents to generate test cases for autonomous cars is previously realized by (Huynh, Gambi & Fraser, 2019), though only a specifically prepared dataset of accident reports by NHTSA was used and these reports contained numerical and very detailed information on how the accident occured. In this study the authors used NLP (Natural Language Processing) to retrieve information from accident reports. These type of datasets can be extremely rare and can take long times to be prepared by humans.

Another idea that inspired our work was the first track of the 2018 NVIDIA AI City Challenge (Naphade, Chang, Sharma, Anastasiu, Jagarlamudi, Chakraborty, Huang, Wang, Liu, Chellappa, Hwang & Lyu, 2018). In this challenge teams of researchers showed their success on analyzing the city's traffic flow from monocular cameras around the city. These cameras are typically in high locations so that they have a wide angle of view. These teams used a combination of artificial intelligence and computer vision methods to analyze the speeds of individual cars in the traffic flow. The works from 2018 NVIDIA AI City Challenge has showed the feasibility of generating traffic analysis using only videos that are taken from monocular cameras.

1.2 Approach

Our approach is much more generalizable, where in this study we only use car accident videos as our only input. Car accident videos are trivial to find as there are a large number of accidents recorded by security cameras and dash cameras that

are mounted on cars. One of the assumptions in this study is that the cameras are mounted on tall poles or buildings, so that it is easier to get a general idea of how the accident happened and there won't be any needing to account for the speed of the vehicle that carry the dash camera. Our approach has three steps: analysis, reconstruction and testing.

The first step is to analyze the accident scene, retrieve and abstract the data from its surroundings. In this step we run the video through an image processing pipeline and extract the speed, 3D position and rotation of vehicles along with the roads, lanes and pavements. This information is then used to reconstruct the vehicles with their respective positions and speeds in the lanes in a simulation environment with realistic physics engine, in the reconstruction step. Then the constructed simulation runs with the computer generated or user defined parameters to finalize the test. The main purpose of this approach is to provide challenging, realistic, configurable and modular test cases for autonomous driving stack developers. A test is typically run with all the vehicles following to their respective routes that is extracted from the video and before the collision happens the control of one the cars involved in the accident is given to an autonomous driving software and the reaction or the ability of the autonomous driving stack to prevent the accident to happen is measured. The car under test (or ego car) can be defined by the user or selected by our software. The moment of control given to the autonomous car can be defined by either time to collision (TTC) or distance to collision (DTC), which is also a parameter in our test case definition.

1.3 Challenges

Our study uses nothing but RGB images to generate full 3D reconstruction of the car accidents. Although this makes it extremely generic and useful, brings various challenges with it. Brief explanations on each of the most critical challenges our work included is as follows:

- **Videos do not contain any information on where the video is taken with respect to the world, with other words the translation and rotation of the camera with respect to the scene is not known:** Our pipeline has no apriori information on the location and rotation of the camera with respect to a fixed point in the scene (eg. a road or a building). The videos

that are used in this study are taken from cameras that has been located in different heights, distances and rotations to the crash scene. This problem causes performance variances between different videos if not handled correctly.

- **Videos do not contain any information on what kind of camera is used:** Thus, our pipeline has no apriori information on the intrinsic and extrinsic camera parameters, including the focal length of the camera, lens, field of view and calibration matrices. Most of the videos used in this study are taken from publicly available sources such as YouTube, therefore contains wide differences in the cameras used. This forced us to develop a novel method to estimate these parameters and calibrate cameras from the available footage.

- **It is hard to evaluate our results, since there is no such dataset and ground truth available for all the information that is used to generate simulation:** Although there exists datasets with 3D tracking information of objects in the scene including camera calibration parameters such as Nuscenes (Caesar, Bankiti, Lang, Vora, Liong, Xu, Krishnan, Pan, Baldan & Beijbom, 2020) and Waymo Open Dataset (Sun, Kretzschmar, Dotiwalla, Chouard, Patnaik, Tsui, Guo, Zhou, Chai, Caine, Vasudevan, Han, Ngiam, Zhao, Timofeev, Ettinger, Krivokon, Gao, Joshi, Zhang, Shlens, Chen & Anguelov, 2020), they have near to no accident footage and there is no simulation implementation done prior to our work using these datasets.

- **The idea and study is unique and there is no prior work on this topic:** This is the first attempt to reconstruct a fully dynamic 3D scene using only one monocular camera source. As of the authors knowledge and capabilities of research, the idea and study is unique, thus there is no previous work on this topic, which brings various difficulties itself.

- **We use cutting edge technologies and software:** Most references and software used in this study are developed after 2019. There is almost no worldwide usage on most of these tools and they are in alpha or beta states. This brings numerous technical difficulties including but not limited to: being the first one find and report bugs, stability issues, breaking changes on the interfaces, custom source builds and modifying the source codes of these tools.

- **There are too many factors to take into account:** There can be various causes of a car accident, including driver's vision being blocked by an external object, a small animal passing by the road and other factors that may or may not be visible in the accident footage. It is one of the most challenging part of our study and the root cause of some of the assumptions.

1.4 Contributions

In the light of this study, our contributions can be summarized as follows:

- We construct a novel image processing pipeline where the output of one neural network is the input of the next neural network. This pipeline includes, 2D object detection, 2D object tracking, 2D object segmentation, and 3D object tracking networks.
- We introduce a novel method to calibrate monocular cameras without any prior knowledge on the scene, and camera's intrinsic and extrinsic parameters. This method features an estimation only using the available footage from itself.
- We introduce a novel method to integrate 2D tracking outputs to improve performance on 3D tracking outputs of CenterTrack (Zhou, Koltun & Krähenbühl, 2020).
- We develop a novel way to define car crash test cases to ease the implementation with different 3D physics simulator options, alternative for using OpenDRIVE (ASAM, 2020a) or OpenSCENARIO (ASAM, 2020b).
- We visualize the car crashes using LGSVL simulator (Rong, Shin, Tabatabaee, Lu, Lemke, Možeiko, Boise, Uhm, Gerow, Mehta, Agafonov, Kim, Sterner, Ushiroda, Reyes, Zelenkovsky & Kim, 2020) and tested Apollo Autonomous Driving (AD) Stack (Xu, Xiao, Miao & Luo, 2020) on various different test cases, generated by our system.
- We implement a modular test case generation executable that given the input car crash video, analyzes the video with the image processing pipeline, reconstructs the 3D scene in LGSVL, runs all the predefined test cases and outputs a detailed report and video result of each step to the user. The executable is reading a configuration file of more than 50 parameters and offers a wide range of configurations in accordance to comply with the users requirements.
- We conduct a user study on how realistic the generated crashes are to further evaluate the results of this study.

1.5 Thesis Outline

Th rest of the thesis is organized as follows:

In Chapter 2, we discuss related work within the context of our study and introduces current state of the art methods in related areas. Where applicable, after each topic there is a table that compares these methods. Chapter 3, describes our approach and system as a whole. Chapter 4, introduces the experiment frameworks and discusses the results. Chapter 5, focuses on issues that could effect the validity and assumptions in our study. Chapter 6, concludes the study with final remarks and finally chapter 7, discusses future work suggestions.

2. BACKGROUND, METHOD SELECTION AND RELATED WORK

Our approach, which is detailed in chapter 3, uses an image processing pipeline that consists 2D object detection, 2D object tracking, 2D object segmentation, camera calibration, speed estimation and 3D multi object tracking steps, to reconstruct car crash videos in a 3D physics simulation. Then an AD stack is integrated to conduct tests and evaluate the performance of the stack on generated test cases. We introduce state of the art methods in each of these steps and compare existing solutions in this chapter.

2.1 2D Object Detection

2D object detection is one of the most studied are in the field of Artificial Intelligence (AI) and computer vision. Although there are still every day improvements in 2D object detection networks and frameworks, there exists a set of industry-proven state of the art solutions to this problems. In the context of our study, including the future work, pedestrians, vehicles and traffic lights are currently the only objects that are important. For this task, networks trained in one of the most common datasets, COCO (Lin, Maire, Belongie, Hays, Perona, Ramanan, Dollár & Zitnick, 2014), were considered and used without any additional training. Although this study does not have any requirements for the methods to be real time, we aim to keep the pipeline as fast as possible to generate more test cases in limited time.

2.1.1 You Only Look Once (YOLO)

YOLO (Redmon, Divvala, Girshick & Farhadi, 2016) is one of the most popular 2D object detection systems and managed to stay as one of the state of the art systems till today with recent improvements on YOLO v4 (Bochkovskiy, Wang & Liao, 2020). YOLO is a unique 2D object detection system that only does one pass on the given image for detecting bounding boxes and class probabilities, contrary to most other systems, which require multiple passes on the image to first detect region proposals and classification. Thus, YOLO is fast and performs extremely well, as only single network is responsible for the detection pipeline, making the network easier to optimize solely based on detection performance.

2.1.2 Faster R-CNN

Evolved from R-CNN (Girshick, Donahue, Darrell & Malik, 2014) and Fast R-CNN (Girshick, 2015), Faster R-CNN (Ren, He, Girshick & Sun, 2017) is a state of the art 2D object detection method that uses a network to make region proposals, as opposed to its predecessors using selective search for the same purpose, making Faster R-CNN multiple times faster and better performing than it's predecessors. More than often Faster R-CNN systems use Residual Networks (ResNets) for feature extraction. ResNets (He, Zhang, Ren & Sun, 2016) have a variety of deep and shallow variants including Resnet-34, ResNet-50, Resnet-101. The core difference between ResNets and other networks is identity shortcuts which lets the network skip layers as desired to avoid performance degrading on deeper networks, allowing researchers to build deeper networks with increasing performances to some degree. Although especially deeper ResNets can slow the object detection system they help the system achieve better overall performances on 2D detection tasks.

2.1.3 Single Shot Detector (SSD)

SSD is also one of the most popular method that only passes the image in one shot to generate multi-object detection (Liu, Anguelov, Erhan, Szegedy, Reed, Fu & Berg, 2016), contrary to Regional Proposal Network (RPN) based methods, such as Faster R-CNN, requiring multiple passes on image. These methods requies a

first pass to generate regional proposals and another pass to detect the objects of respective proposals as detailed in subsection 2.1.2.

Although the original SSD can be used as a standalone method, today it is also usually used with a different network integrated into SSD architecture to optimize detection performance for different use cases, the most commonly used network being MobileNets (Howard, Zhu, Chen, Kalenichenko, Wang, Weyand, Andreetto & Adam, 2017; Sandler, Howard, Zhu, Zhmoginov & Chen, 2018). MobileNets are a special type of Convolutional Neural Network (CNN) and are known for being extremely fast, as they use a special depth-wise separable convolutions, which makes them remarkably lightweight in terms of computational complexity and parameter count.

Even though SSD, depending on the network that is used along with, usually fall short in accuracy rankings against Faster R-CNN and YOLO, it is much faster, which makes it widely used on devices with limited computing capabilities.

2.2 2D Object Tracking

2D object tracking is also an important part of this study, as well as many AI researches. There are many cases where detecting the class of an object on a single frame is not enough and the objects needs to be assigned with a unique identifier. This problem is specifically important in video processing and the cases where the object itself is crucial rather than the type of it. There are various different solutions to object tracking problem, this section discusses two of these tracking methods.

2.2.1 Simple Online and Realtime Tracking (SORT)

SORT (Bewley, Ge, Ott, Ramos & Upcroft, 2016) is one of the state of the art methods of the earlier generation of 2D object tracking algorithms, that revolutionized the field by ranking the best open source implementation in one of the most competitive benchmark, MOT15 (Leal-Taixé, Milan, Reid, Roth & Schindler, 2015) with a simple approach. Due to the ease of implementation and use SORT is still considered to be one of the widest used algorithms. In addition, a stronger and more

recent version of SORT exists, that is called Deep SORT, which simply improves the original implementation via information derived from deep appearance descriptors. SORT utilized the visual features from an object detection algorithm along with a Kalman Filter for state estimation and Hungarian algorithm for matching and clustering. SORT is unable to identify re-entering objects which happens quite often in our studied videos, which was the reason to eliminate this algorithm from our image processing pipeline.

2.2.2 TC Tracker

TC Tracker (Tang, Wang, Xiao, Zheng & Hwang, 2018a) utilizes a combination of visual and semantic features typically found from 2D object detection as a backbone of the tracking algorithm. It then calculates a loss function with a model that is based on histograms, which is used to find associations between tracklets in the clustering step. TC Tracker proved its capabilities in similar single camera footages that are taken from high location. With this proven performance record, we have chosen to utilize this method in 2D object tracking part of our image processing pipeline.

2.3 2D Object Segmentation

2D Object Segmentation is used extensively to analyze objects more accurately. Object detection algorithms can only insert bounding boxes, which can not take arbitrary shapes and faulty in representing an exact location or area that object covers. Segmentation algorithms however use masks that can take any shape and much more precise on representing 3D objects. In the context of our study we use object segmentation to serve as a base of camera calibration using masks of vehicles and traffic lines for calibration as explained further in sections 3.1.3 and 3.1.4.

2.3.1 Mask R-CNN

Mask R-CNN (He, Gkioxari, Dollár & Girshick, 2020) is considered to be one of the methods that revolutionized 2D object segmentation by using a simple methodology that outperformed every competitor algorithm in COCO benchmarks (Lin et al., 2014) in the time of publish. Mask R-CNN extends the capabilities of Faster R-CNN with segmentation masks with a fully convolutional network (FCN) (Long, Shelhamer & Darrell, 2014), that is passed through each object detection and classification as an overhead. Although Mask R-CNN is now considered to be a baseline algorithm in object segmentation, there other methods exists such as In Place Activated BatchNorm, that outperforms Mask R-CNN in various datasets.

2.3.2 In Place Activated BatchNorm

In Place Activated Batch Normalization (InPlace-ABN) (Bulo, Porzi & Kontschieder, 2018) is a state of the art method that is specifically developed for traffic datasets by Mapillary Research. InPlace-ABN is the best performing algorithm in Mapillary Vistas (Neuhold, Ollmann, Bulo & Kontschieder, 2017), Cityscapes (Cordts, Omran, Ramos, Rehfeld, Enzweiler, Benenson, Franke, Roth & Schiele, 2016), KITTI (Geiger, Lenz, Stiller & Urtasun, 2013) and ScanNet (Dai, Chang, Savva, Halber, Funkhouser & Nießner, 2017) benchmarsk by introducing a unconventional new layer called In-Place Activated Batch Normalization, instead of using the regular method of BatchNorm and activation layers after each other. InPlace-ABN is a new layer plugin that combines BatchNorm and activation layers in a unique way withm performance increase and saves up to 50% GPU memory, which enables the use of deeper and heavier networks with this method. Since this algorithm is still performing in the top of many benchmarks and was able to handle the videos that are taken from highly located cameras within our tests, we have chosen to use this method.

2.4 Camera Calibration and Speed Estimation

Most of the related works that we have considered does integrate or relate camera calibration and speed estimation processes, thus these two parts will be considered in one section. Camera calibration is the process of estimating Intrinsic and extrinsic camera calibration matrix that is required by various methods that are used in this study including, depth estimation for speed estimation and 3D Multi Object tracking. Speed estimation is the process of estimating the speeds of dynamic objects in length per seconds rather than pixels per seconds. This is considered to be hard problem to overcome by using monocular camera images without the help of additional sensors, since the depth estimation and camera calibration brings accuracy problems. This problem is mostly studied for NVIDIA AI City Challenge's Track 1 in 2018 (Naphade et al., 2018), which was about anlyzing the traffic flow within CCTV footages around the city cameras, making it contain similar footages that we also use in our studies.

The winner of Track 1 from University of Washington (Tang, Wang, Xiao, Zheng & Hwang, 2018b), requires a camera calibration method that requires a human intervention that labels two parallel traffic lines, from which the algorithm finds the ground and estimates the camera calibration matrix with small constraints. Speed estimation is then done by simply extracting how much pixels did the tracked vehicle moved between each frame and dividing it by time passed to achieve px/s values. The algorithm is completed by converting px/s values to m/s tracked vehicles using a depth estimator that uses camera calibration matrix.

One of the best performers in the same challenge, University of Maryland (Kumar, Khorramshahi, Lin, Dhar, Chen & Chellappa, 2018), can perform speed estimation without any camera calibration by assuming the road and camera are parallel and locating a vanishing point through the road for image rectification. For speed estimation they use a combination of affine transformation within rectified image and Kalman filter with scaling that is performed by estimating the distances from Google Maps. This method was not applicable in our study since we don't have prior knowledge of the images and cameras.

We use a similar approach for speed estimation with University of Washington with modified constraints , and introduce a novel method for camera calibration that does not require human intervention and automate the whole process.

2.5 3D Multi Object Tracking

3D Multi Object Tracking (MOT), is one of the recently advanced areas in the field of AI. 3D MOT algorithms are mainly developed to be used with autonomous vehicles, where these algorithm estimate 3D poses (all 6 degrees of freedom including 3 Axis for translation and 3 axis for rotation) of the objects. Being computationally heavy and a complex problem to solve, 3D MOT algorithms typically use a combination of multiple sensors including 3D LIDARs and multiple depth and monocular cameras. There are only a few open source implementations of 3D MOT, using a single monocular camera. Identifying 3D poses of objects with only one single monocular makes these algorithms fit into our study, where we use rotation estimation solely based on 3D MOT outputs. We discuss two popular algorithms that are highly ranked in popular datasets.

2.5.1 CenterTrack

CenterTrack (Zhou et al., 2020; Zhou, Wang & Krähenbühl, 2019) is one of the state of the art algorithms, being the best performer in KITTI (Geiger et al., 2013) and MOT17 (Dendorfer, Osep, Milan, Schindler, Cremers, Reid, Roth & Leal-Taixé, 2021) benchmarks the time it is published. CenterTrack uses a more traditional approach of tracking the central points of interested objects in contrast with most tracking algorithms, that use tracking-by-detection approach, where they are usually complimented by a detection algorithm and uses clustering or different neural networks to achieve tracking with the prior knowledge of objects. Tracking only the central points of moving objects, makes CenterTrack faster than most networks. Our study uses a modified version of CenterTrack that provides a prior knowledge of object locations which increase the accuracy performance even further.

2.5.2 DEFT

DEFT (Detection Embeddings for Tracking) (Chaabane, Zhang, Beveridge & O'Hara, 2021), is another rare tracker that does not use tracking-by detection

methodology. Instead, DEFT uses an LSTM for motion constraints and completes the joint detection and tracking by comparing the appearances of objects only by using a single monocular camera. Similar to what this study applied to CenterTrack, an advantage of DEFT is that it can accept a detection algorithm to further increase accuracy. Down side of the way it applied is it requires retraining DEFT alongside the object detection algorithm, which decreases the modularity of such implementation. DEFT is not chosen for this study, as it is simply performing worse then CenterTrack in the benchmarks and in our configuration. The best performing DEFT uses CenterNet (Duan, Bai, Xie, Qi, Huang & Tian, 2019), which uses a similar approach of CenterTrack for object detection tasks.

2.6 3D Physics Simulation

Physics simulators are usually referred to computer software that includes a realistic physics engine and preferably a photorealistic graphical user interface that represent visuals. Collisions are modeled with fast collision checking libraries under the physics engine in the background and realistic visuals are rendered in the front end. In the context of our study we use physics engines to realistically recreate the accident and integrate autonomous cars to conduct our use case studies.

2.6.1 LGSVL

LGSVL (Rong et al., 2020), is a simulator developed in Unity (Technologies, Technologies) game engine by electronic giant LG's Silicon Valley Lab and features photorealistic scenes, fast 3D LIDAR simulation, easily editable maps and vehicles. LGSVL provides a modular interface to add and configure a variety of sensors and provides off-the-shelf support for two of the most popular publicly available autonomous driving stacks, Autoware and Baidu's Apollo. Asides from Autoware and Baidu users can integrate any autonomous driving stack via supported communication bridges, namely ROS, ROS2 and CyberRT. LGSVL has an extensive Python Application Programming Interface (API) that can spawn dynamic and controllable objects such as pedestrians and vehicles, along with a collection of useful static objects such as traffic lights. Our previous experience with Unity and the modular way

that LGSVL is written are the main reasons this study has selected to use LGSVL. Figure 2.1 shows the high level communication architecture between autonomous driving stacks and LGSVL.

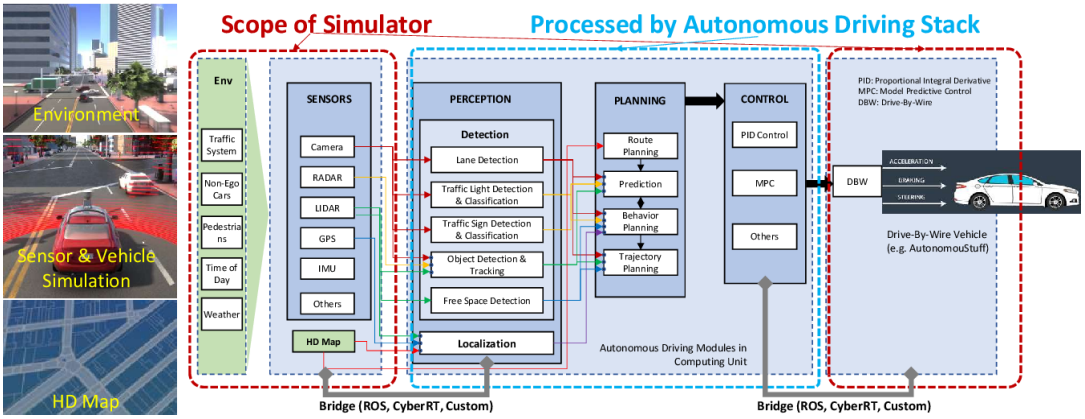

Figure 2.1 LGSVL integration with Autonomous Driving Stacks

2.6.2 CARLA

CARLA (Dosovitskiy, Ros, Codevilla & Antonio, 2017) is another autonomous driving simulator that is developed in Unreal Engine (Games, Games). Similar to LGSVL, CARLA features photorealistic scenes, map generation with OpenDrive, extensive sensor suite and ROS integration, as well as a scenario runner to easily integrate and run test cases.

2.6.3 Gazebo

Gazebo (Koenig & Howard, 2004) is a popular robotics simulator that provides a selection of some of the most popular physics engines such as ODE (Smith & others, 2005), Bullet (Coumans, 2015) and DART (Lee, X. Grey, Ha, Kunz, Jain, Ye, S. Srinivasa, Stilman & Karen Liu, 2018) for dynamic interactions. The ability to change and integrate different physics engines gives Gazebo an edge against its competitors, however Gazebo is typically considered to be less photorealistic than CARLA and LGSVL, which are built on top of game engines. Although, gazebo provides a large selection of sensors and extensive integration with ROS and ROS2 it does not have a direct support for spawning random traffic, and other dynamic

objects that can be considered to be essential for autonomous car simulation.

2.7 Autonomous Driving Software Stacks

Although there are great advancements in autonomous driving technologies recently, most of the work done are commercial and closed source. In this section we discuss two popular open source methods that are available to use as autonomous driving stacks. Autonomous driving stacks represents the software component of self driving cars. They are usually integrated with high end CPUs and GPUs along with a collection of sensors. All the afore-mentioned simulators can provide realistic sensors that are used in self driving cars with environmental noise features. So the selection for autonomous driving stack here is solely based on software performance and integrability. In this study we use autonomous driving stacks to test our test case generation framework and demonstrate the usability of our software.

2.7.1 Autoware

Autoware (Kato, Tokunaga, Maruyama, Maeda, Hirabayashi, Kitsukawa, Monrroy, Ando, Fujii & Azumi, 2018), is developed completely open source by efforts of numerous companies together. It is fully compatible with ROS and ROS2. Supporting ROS and ROS2 Autoware is compatible with almost all simulators in the current market. Autoware is modular and ships with multiple software stacks for each driving module, such as perception, control and planning. This provides Autoware with a flexibility to swap each module with custom algorithms. On the other hand this flexibility causes Autoware harder to configure and use as is. Autoware uses RViz (ROS Visualization) (Kam, Lee, Park & Kim, 2015) for data visualizations, which is the default visualization tool for current ROS and ROS2 distributions.

2.7.2 Apollo

Apollo (Xu et al., 2020; Zhu, Ma, Xu, Guo, Cui & Kong, Zhu et al.) is a commercial AD stack of Chinese tech giant Baidu. Although being commercial and actually used on roads, Apollo is open source and even easily integrable to custom hardware. There are different versions of Apollo available, though we are only considering Apollo 5.0 in the context of this study, since 6.0 was published later into our development progress. Apollo uses its own communication protocol, namely CyberRT for intra-process communications, simulation and hardware bridges. Although, CyberRT is not supported on all simulators and is a new protocol with steep learning curve, LGSVL, provides an easy to use API to integrate with CyberRT. Apollo's software is designed as a modular framework and one can easily swap different modules such as localization, perception and routing with custom modules. Apollo is shipped with a verbose UI, namely Dreamview. Apollo is robust in lane following and responsive to sudden changes in environment. In our tests we found that Apollo to be easier to integrate and more ready and robust than Autoware in our configuration, thus used it for running generated test cases.

2.8 Software Selection

Table 2.1 shows the final selections of software and their use cases that are discussed in this chapter.

Table 2.1 Software selections in this study

Use Case	Software Selection
2D Object Detection	YOLO
2D Object Tracking	TC Tracker
2D Object Segmentation	In Place Activated Batch Norm
Camera Calibration	Custom
Speed Estimation	UW NVIDIA 2018
3D Object Tracking	Modified CenterTrack
Simulator	LGSVL
Autonomous Driving Stack	Apollo 5.0

3. Methods

Our ultimate goal with this study is to generate realistic test cases for autonomous vehicles in an automated manner, to fill the lack of generating such test cases without any human intervention. Although, (Huynh et al., 2019) seems to automate this process, the datasets used are specifically prepared by NHTSA with humans in the loop.

We wanted our approach to be generic, scalable and usable within research communities and industry. Thus, we are presenting a unique way to generate these test cases, using only videos taken from monocular cameras around the world without any prior knowledge in scene, crash or cameras. This increases the significance of our study to an impressive potential of being capable of generating test cases from every car crash video in the world. The analysis and scene reconstruction steps has also potentials to be used in a wide range of applications including security, robotics, film casting and law.

Our approach to the problem of constructing a 3D scene, and generating test cases of AD stacks only using series of 2D images, has 3 main steps, which are:

- Analysis
- Scene Reconstruction
- Test Case Generation

3.1 Analysis

In the analysis step, we construct a special and rather complex image processing pipeline that only takes one input, which being the original car crash video itself

in a common video format such as mp4, flv or mkv. The pipeline outputs all the necessary data to reconstruct the 3D scene.

The image processing pipeline consists of three different neural network and six internal steps in total which can be listed as follows:

- 2D Object Detection
- 2D Object Tracking
- 2D Object Segmentation
- Camera Calibration
- 2D Speed Estimation
- 3D Object Tracking

The full input and output configuration of the pipeline can be seen in Figure 3.1.

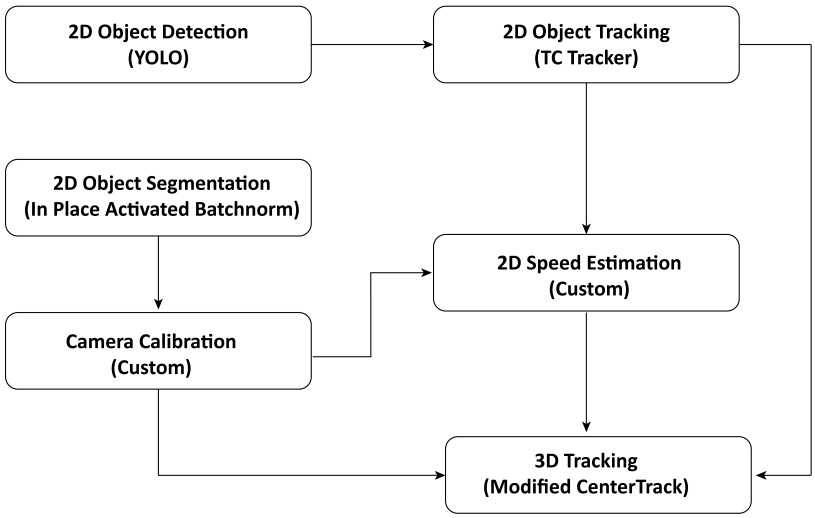

Figure 3.1 Image processing pipeline used in the analysis step

3.1.1 2D Object Detection

The first step is 2D object detection where a CNN, YOLOv4 is used for classification of objects. YOLO can classify and mark the objects with 2D bounding boxes, classification labels, detection frame IDs and confidence values. We retrained YOLO's

pre-trained COCO model on a custom dataset that includes a mix of two available datasets, namely Jakarta Smart City Dataset and KITTI. The motivation behind retraining the network is to achieve better performance on the videos that are taken from high locations. An example output frame from this step can be seen in Figure 3.2.

Jakarta Smart City Dataset (Caldeira, Fout, Kesari, Sefala, Walsh, Dupre, Khaefi, Setiaji, Hodge, Pramestri & Imtiyazi, 2020) contains 700GB of raw video footage from 7 CCTV cameras around the Jakarta City to better analyze traffic behaviour around the city. Although the dataset was not labeled, we randomly extracted 2000 images and labeled from this dataset to increase our detection performance on CCTV camera footages.

KITTI (Geiger et al., 2013) is one of the most popular datasets for autonomous cars and consists over 15000 labeled 2D and 3D test and training images. KITTI has been used to analyze the performance of current state of the art methods including (Weng, Wang, Held & Kitani, 2020) and (Zhou et al., 2020).

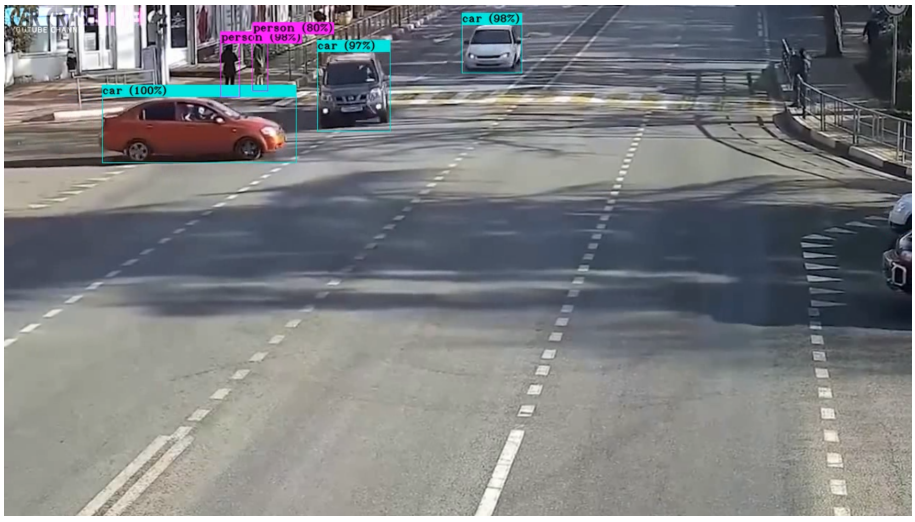

Figure 3.2 Output of YOLO's 2D Object Detection in one of the analyzed crashes

3.1.2 2D Object Tracking

The second step in the pipeline is 2D object tracking. To analyze a car crash one needs to know more than the labels of the objects. As our study analyzes attributes for every unique object in the videos, a unique id for each object needs to be assigned. The unique id assignments are 2D tracking neural networks. For 2D

tracking, we are using TC Tracker (Tang et al., 2018b), a proven clustering based tracking method that runs on Matlab. This method was used for University of Washington's 2018 NVIDIA AI City Challenge implementation with great success. The input for this algorithm is the output of 2D detection step which are the 2D bounding boxes, classification labels, detection frame IDs and confidence values. The outputs contains one extra column which is the unique id of the object. A sample output frame from 2D tracking step is visualized in Figure 3.3.

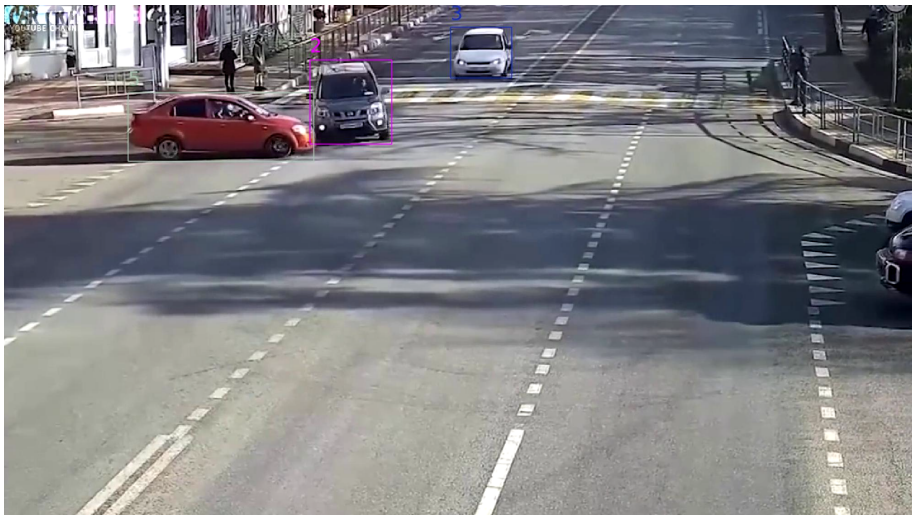

Figure 3.3 Output of TC Tracker's 2D Object Tracking in one of the analyzed crashes

3.1.3 2D Object Segmentation

The third step in the pipeline is 2D object segmentation. Object segmentation is especially important for the scalability of this study and allows many of the potential future works to be feasible. Object segmentation is used to detect the vehicles and roads in masks. Masks are more precise representation than bounding boxes to, as it can represent all the edges correctly. The detections on this step are used as a base for camera calibration in the analysis step and mapping the points between road from the video and map of the simulation in the scene reconstruction step in Section 3.2. The image segmentation algorithm used in our analysis is a pre-trained version of In Place Activated Batchnorm (Bulo et al., 2018) network using WideResNet-38. An output of 2D segmentation from the same video can be seen in Figure 3.4

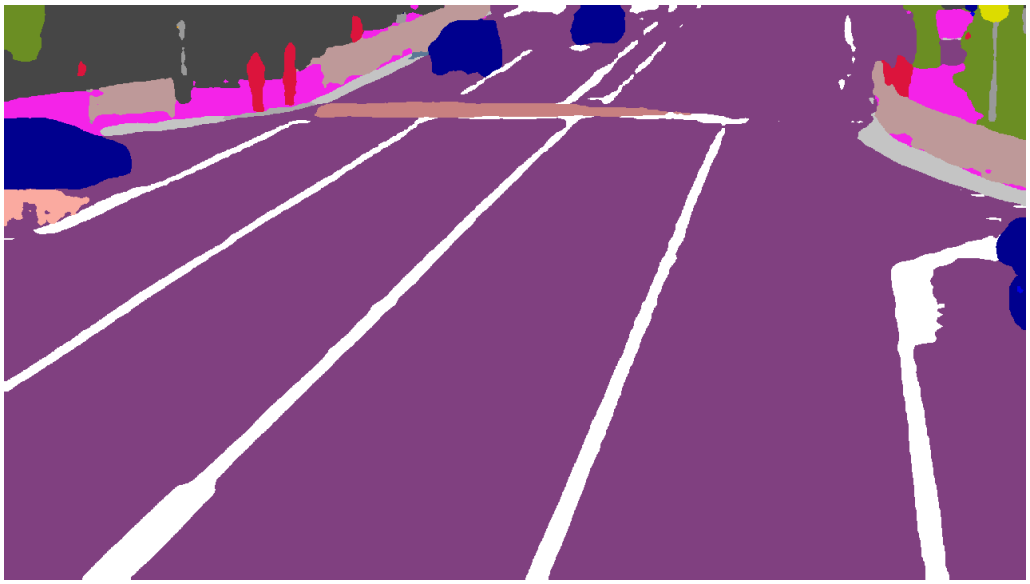

Figure 3.4 Output of 2D Object Segmentation in one of the analyzed crashes

3.1.4 Camera Calibration

The next process in the pipeline is camera calibration. The camera calibration is executed on the output images of 2D object segmentation process. The single frame on the video with the most cars detected is selected by default with an option given to user to select any frame desired. In the case where there are more than one frames that contain the same number of maximum cars detected, the first one to appear is selected.

This process starts with, selecting a car that is considered aligned with the traffic lines for doing the calibration. The car is selected via an additional pre-processing step that finds the relative angle between all the cars and all the traffic lines. The smallest relative value is selected.

Then an affine transformation is applied to image to find the rotation between the camera and the car. The affine transformation is applied such that the car is aligned with vertical axis of the image. In Figure 3.5, the car with id “3” is selected by the calibration program and the image is rotated with affine transformation as such. The rotation amount from the affine transformation is directly passed to the scene reconstruction step.

Figure 3.5 Output of 2D affine transformation in one of the analyzed crashes

The full 3D calibration needs at least 2 points in each of the 3D Cartesian axes. These points are selected from the chosen car's furthest points of each axes, namely, from side (Y axis), back (X axis) and height (Z axis). Then the pixel values are compared with average sedan car length, width and height to find pixel to meter ratios which are also directly passed to scene reconstruction step. The average values that are used in this study are taken from 2020 model Renault Megane, which as follows:

- Width: 1814mm
- Length: 4356mm
- Height: 1436mm

Lastly, the full camera calibration containing intrinsic and extrinsic parameters is completed applying the open source implementation of (Lee, Lee & Hwang, 2013) by using cars instead of people, which also uses these 6 points for self-calibration.

The assumptions in this process are as follows:

- The video contains traffic lines.
- At least one car is close to parallel (within 15 degrees) with a traffic line.

3.1.5 Speed Estimation

The fifth step in this pipeline is speed estimation. The inputs to this step are the outputs of 2D tracking step and the full camera calibration from the relevant process. The open source speed estimation implementation of (Tang et al., 2018b) is used with pre-defined minimum and maximum speed constraints. The method essentially calculates how much each vehicle has travelled using the 2D tracking and calibration information and takes the derivative of position over time to estimate the speed. The method also applies a projection correction, which uses the closer objects as a base to estimate farther object's speeds to decrease estimation errors caused by the depth. Figure 3.6 show an image from the output of speed estimation process. The speeds are labeled as a unit of km/h.

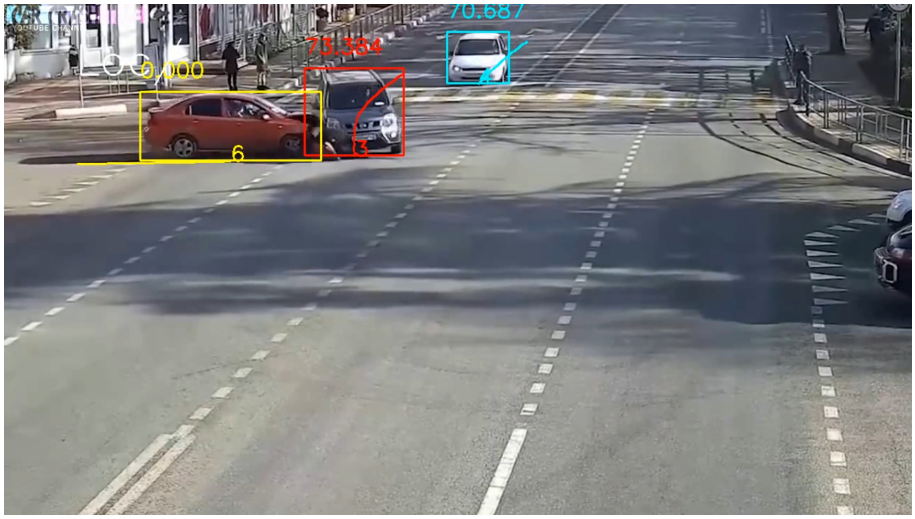

Figure 3.6 Output of speed estimation in one of the analyzed crashes

3.1.6 3D Object Tracking

The final step of the image processing pipeline is 3D tracking. Although the previous outputs of the pipeline is sufficient to reconstruct a scene, the tests showed that the car's rotation errors are too high to ignore and be compensated. These tests were conducted such that the rotations of the cars were to be calculated using a quintic G^2-spline trajectory generation from the 2D spline path that is following the center of the car's bounding boxes from 2D tracking process (Piazzi & Guarino Lo Bianco,

2002), with the form of equation 3.1.

$$\theta(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3 + a_4 t^4 + a_5 t^5 \tag{3.1}$$

The trajectory generation does not account the camera calibration, thus causes high rotation errors depending on how angled the camera is looking. This problem is solved by this extra 3D object tracking process which accounts for camera rotations and thus finds the rotations, namely the yaw angle, of the cars more accurately.

3D multi object detection (MOT) is common to be done using a combination sensors, being primarily developed for autonomous car applications. Using only video images to achieve a full 3D scene reconstruction limited our study to be able to use the camera only methods, which is not the usual case and performs poorer than the multi sensor methods understandably. Out of all camera only methods we have experimented with we found CenterTrack (Zhou et al., 2020) to be the most promising by using the object centers with the combination of camera calibration (Yin, Zhou & Krähenbühl, 2020; Zhou et al., 2019) method to be the most promising, which is also the best performing camera only method in Waymo Open Dataset (Sun et al., 2020).

The method is essentially an end-to-end algorithm, which also derives the 2D object detection, tracking and speed estimation itself. However, as an end-to-end solution CenterTrack performed poorly, finding the vehicles. This is mainly due to CenterTrack being trained in NuScenes dataset (Caesar et al., 2020), which only contains footages from car dash cameras, where this study mainly contains static footages taken from CCTV cameras. Due to limited GPU resources and the lack of 3D MOT datasets that contain CCTV footages, instead of training this network, our study modifies it such that it takes the 2D tracking and speed estimation information from the previous processes of this pipeline along with the already used camera calibration matrices. The network is informed such that it now assumes that there is a unique object with a specific speed in a location exists, and generates the 3D tracking estimation based on this initial estimate. This increased the detection and tracking performance of this network and made it useful for our study. Although the network still generates full 3D tracking estimation with 3D bounding boxes and 3D speed estimations, we only use the rotation angle outputs of this network, since the performance of 2D tracking, and speed estimation algorithms were already enough for an accurate estimation of car locations in scene reconstruction step. Figure 3.7 shows a sample image that contains 3D bounding boxes, newly assigned unique ids

and speed estimation (with pink arrows) of the 3D tracking output.

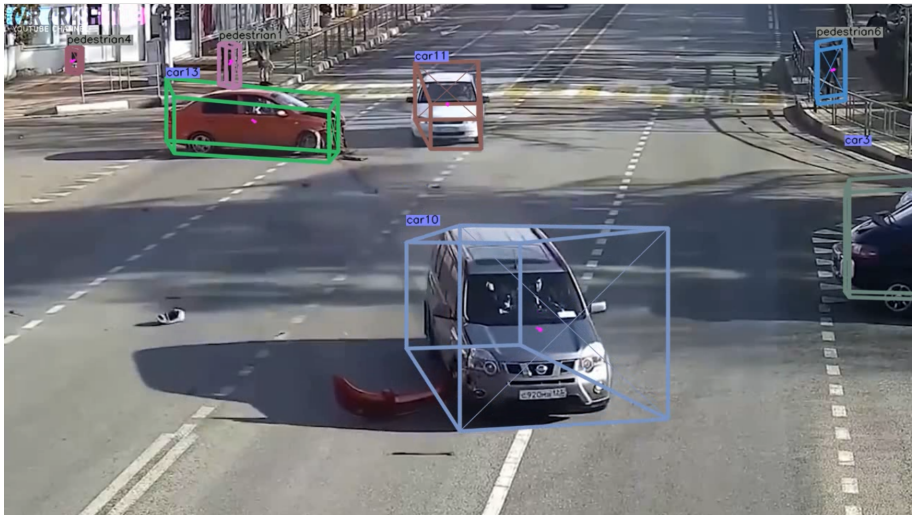

Figure 3.7 Output of 3D tracking network in one of the analyzed crashes

3.1.7 Outputs

The final output of the image processing pipeline consists of 2 files, one file including all the information necessary to locate and move the objects in the scene, and another file containing a one line calibration information to make pixel to meter transformations and define necessary rotation matrices. In our study we define the format of the former file as follows, which is a redefined version of the format being used in Multi Object Tracking (MOT) Challenge (Milan, Leal-Taixe, Reid, Roth & Schindler, 2016) as follows:

$$< class >, < frame >, < id >, < bb\_left >, < bb\_top >, < bb\_right >, < bb\_bottom >,$$

<speed>, <confidence>, <rotation>

and the definitions for these elements are stated as follows:

- **class:** Numerical representation of detected class of the object (such as 1 for car, 2 for pedestrian etc.)

- **frame:** The frame number (counting from 1) of the data in this line.
- **id:** Unique tracking id of the detected object
- **bb_left:** The top left point's X value (in pixels) of the 2D bounding box, that the object is detected.
- **bb_top:** The top left point's Y value (in pixels) of the 2D bounding box, that the object is detected.
- **bb_right:** The bottom right point's X value (in pixels) of the 2D bounding box, that the object is detected.
- **bb_bottom:** The bottom right point's Y value (in pixels) of the 2D bounding box, that the object is detected.
- **speed:** The speed of the object in km/h.
- **confidence:** Probability value of the object's presence between 0 and 1, according to 3D tracking algorithm.
- **rotation:** Heading angle of the object with respect to the camera in radians.

A sample portion of an output file can be seen in listing 3.1.

```
1 1 125 3 428 85 553 205 77.370 0.87 -1.72138520249562
2 1 125 4 626 37 713 108 73.160 0.84 -1.4583021104770997
3 1 125 6 186 125 443 220 7.441 0.91 -0.1724041104169461
4 1 126 3 425 88 557 209 75.123 0.87 -1.7215194937681721
5 1 126 4 624 38 711 110 71.883 0.83 -1.4501205016055827
6 1 126 6 189 125 445 220 0.000 0.92 -0.13362234296247422
```

Listing 3.1 A sample from the data output file of an analysis step

The calibration file is a one line file which contains the following information:

$$< x\_px\_to\_meter >, < y\_px\_to\_meter >, < z\_px\_to\_meter >, < rotation >$$

where:

- **x_px_to_meter:** A calibration parameter to transform pixel values in the data output file to meters in X axis.
- **y_px_to_meter:** A calibration parameter to transform pixel values in the data output file to meters in Y axis.

- **z__px__to__meter:** A calibration parameter to transform pixel values in the data output file to meters in Z axis.
- **rotation:** The amount of rotation that should be applied to data output around Z axis with respect to the scene in radians.

An example of such calibration file can be seen in listing 3.2.

```
0.26179 55.9411764706 0.022222 10.3333333333
```

Listing 3.2 A sample calibration file of an analysis step

The outputs of the analysis step forms the inputs of the scene reconstruction step.

3.2 Scene Reconstruction

The scene reconstruction step includes spawning vehicles and other objects read from the input, creating the map, traffic lights, weather and all the related environment in a physics simulator. For the implementation simplicity this study only models the vehicles in the 3D Physics simulator to demonstrate the capabilities of our image processing pipeline.

The simulator chosen for this task, LGSVL, contains different maps, vehicles and allows users to modify the environment variables, such as changing the color of traffic lights and weather through a Python API. However the maps are loaded as 3D meshes and normally constructed through a detailed Unity User Interface, that can be found in the source built version of the simulator. LGSVL and no other simulator to our knowledge does not support automatic map generating through road definition files. Thus, although part of the environment such as roads and traffic lights are analyzed in analysis step, the simulation only includes the vehicles and uses the same map for every video.

In this step, the input speeds and locations are read from the outputs of analysis step and changed to a format available from the Python API, trajectories are piped through a Savitzky–Golay filter (Merkle & Discher, 1964) and motions are limited in accordance with the velocity, acceleration and jerk limits of the vehicles. Savitzky–Golay filter are used to smooth the this time-series (in this case frame-series) data to achieve smoother motion instead of the jiggly motion output from the tracking. A sample output from the Savitzky–Golay filter applied to a tracked vehicle's

X and Y positions are given in Figure 3.8.

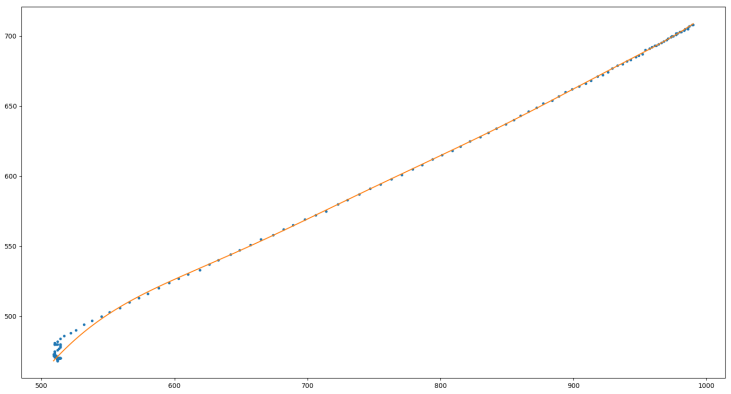

Figure 3.8 Output of Savitzky–Golay filter applied to the tracking data of a tracked vehicle

Then the camera is placed in an automatically determined location in accordance with the camera calibration matrix, estimated depths of the cars are pixel to meter ratios.

One last important point in the scene reconstruction step is matching the actual roads in the accident and the simulator. The roads may not match exactly with the accident and Borregas Ave map, that is used for the simulation. To automate that process, the roads in the simulation map are hardcoded to be matched with the road on the video. Each lane in the simulator is assigned a unique id and direction with a spawning position. The vehicles on the accident videos are matched to these hardcoded spawning positions to achieve a realistic mapping between the video and simulator map. Figure 3.9 shows an empty Borregas Ave map from LGSVL, that is used for all the scene reconstructions.

Figure 3.9 Borregas Ave in LGSVL

A rotation and translation is then applied to every tracked car's position in accordance with the map matching and pixel to meter ratios taken from the analysis step. The rotation and translations are applied in the forms of a simple 2D translation and rotation matrices, which can be seen in 3.2 and 3.3.

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.3}$$

All the translation and rotations are applied relative to a reference vehicle. This so called "reference vehicle" is the same vehicle that is selected in the 2D object segmentation step with the lowest relative angle difference to the closest traffic line. The z position of the vehicles are ray casted to the closest ground point in the map which is another assumption that the accident happened in a flat surface.

In the ray casting implementation our study uses the readily available raycasting method through LGSVL's Python API. The method imitates a car's position as a light being refracted through hitting an object. The car here is simulated as a light source and the object is simulated the closest road.

With all the known positions, speeds and time steps for each tracked vehicle, the

vehicles are placed into their spawn locations and spawned in the time step where they are appeared in the video. Due to a reported known bug in the LGSVL simulator, these vehicles are visualized floating, but not included in physics. Figure 3.10 shows a bird-eye 2D view of the tracked vehicles after the transformation process and before the full 3D scene reconstruction. Here the vehicles are represented as blue arrows, where the starting point of the arrow represents the center of a car and the direction of the car is aligned with the actual yaw angle. The tracked path is represented as red dots and the unique IDs are placed on top of each vehicle.

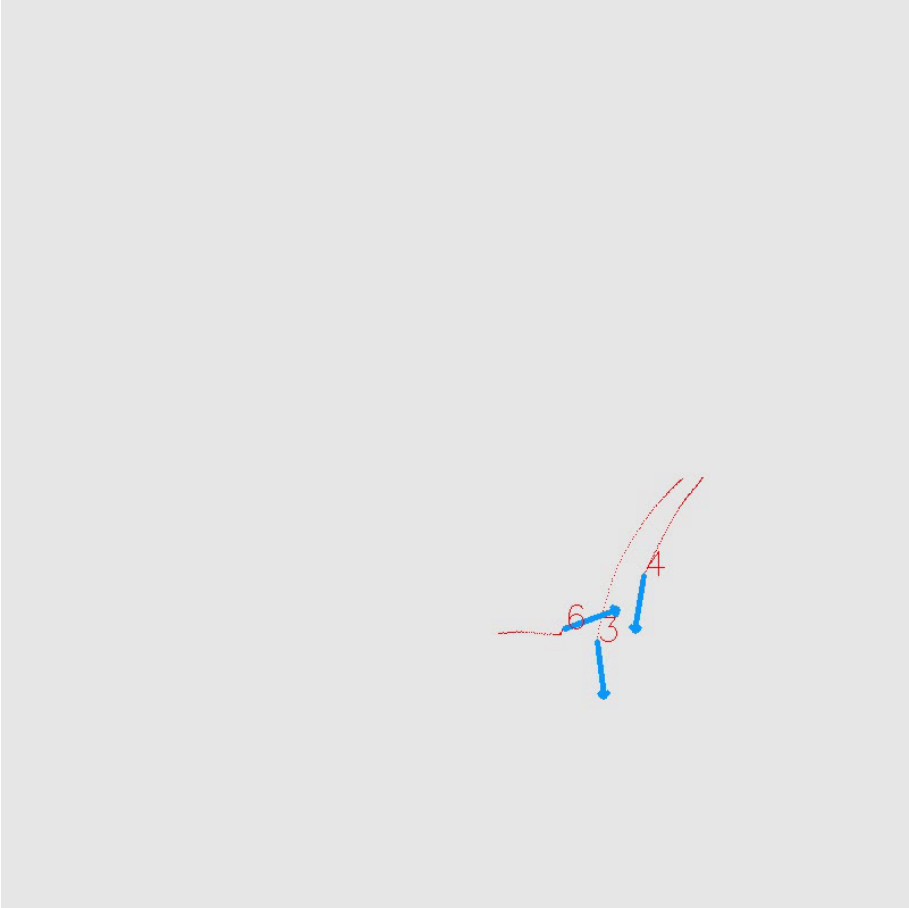

Figure 3.10 2D bird-eye view of tracked vehicles

With all these steps including motion filtering, camera placement, spawning vehicles and map matching the scene can be fully reconstructed. Figure 3.11 shows a side by side comparison between the actual accident and the reconstructed accident in LGSVL simulator by using Borregas Ave Map. Note that the color of the cars are ignored in this simulation.

(a) Original accident

(b) Recreated accident

Figure 3.11 Side by side comparison of the actual accident and the recreated accident

3.3 Test Case Generation

After the simulator is reconstructed the vehicles that are involved in the accident are swapped with autonomous cars one by one and various tests are done for each swapped vehicle.

At this step, the video is already analyzed and the scene is reconstructed in LGSVL. All the tester should do is to conduct the test by defining a few parameters and start the automated tests. In these tests, there are two types of vehicles: non playable characters (NPCs) and ego cars (or car under test). This study assumes that only one vehicle is an ego car and the rest of the scene is replayed by NPCs. NPCs follow their usual trajectory that is already derived in analysis and scene reconstruction steps. However after their trajectories are over NPC vehicles are ordered to follow the closest lane via Python API, so that an analysis can be done in the case accident is avoided by the ego var. A parametric post accident time is added to the simulation runs for after accident analysis. In the simulation, NPC vehicles are selected to be random Sedan cars. Ego car can be selected by the user and typically is provided by the autonomous driving stack to the simulator.

Tests are automatically stopped after a crash is detected between the ego car and any of the other vehicles. Although the test results are always either a pass or a fail, the test conductor can select to retrieve valuable data, including various raw or processed data from sensors, throttle and odometry to further analyze the reaction of the ego car.

Technically the ego car is the autonomous vehicle, and it is expected from the test conductor to provide an autonomous driving stack. LGSVL's modular interface allows one to swap and test different autonomous driving stacks with ease while providing support for ROS, ROS2 and Cyber RT with modular and extensive API. By default our study controls the ego car with Apollo 5.0 (Xu et al., 2020; Zhu, Ma, Xu, Guo, Cui & Kong, Zhu et al.), the Chinese tech giant Baidu's autonomous driving stack. Apollo is publicly available, capable of thoroughly following it's given route, extensible and scalable with their API and Cyber RT protocol. LGSVL provides a bridge to communicate with Apollo, where LGSVL provides the data from simulated sensors, Apollo uses a combination of different software stacks including navigation, localization, mapping, perception to calculate the necessary throttle and control the ego car. Although typically the target location is given by the users in autonomous vehicles, this study automates this process, such that the target waypoint is automatically calculated by forward predicting the tracked vehicle's desired position in time via curve fitting and provided to Apollo's internal stacks. The target waypoint of the autonomous vehicle is selected as the closest lane of the curve fitting location using a pre-defined number of seconds after the accident would have happened by default and defined as a parameter. The accident time here is defined as the first collision between any of the vehicles in a dry run that is done without an ego car.

As mentioned above out of two popularly available autonomous driving stacks, we use Baidu's Apollo. Apollo comes with a ready-to-use interface through Python API of LGSVL, which was of one the main reasons why this study uses it, asides from it's stability over the other popular AD stack, Autoware Kato et al. (2018). The autonomous cars of Apollo in LGSVL contains the following sensors:

- GPS / INS
- 3D LIDAR
- Radar
- IMU
- Colored Monocular Cameras
- Stop Line Sensor

In addition to these sensors we have included a 3D Ground Truth sensor and a Traffic signal sensor that retrieves ground truth information from the simulator. 3D Ground Truth sensor pipes a collection of 3D bounding boxes that represents the nearby vehicles that are in a certain range of the autonomous vehicle. Traffic signal sensor automatically recognizes the color of the nearby traffic lights and reports it to the vehicle. Both sensors are used instead of the unavailability of Apollo's Perception system that requires very high-end GPUs to work on, which was unavailable to our study due to limited resources. A full view of Apollo working in LGSVL can be seen in Figure 3.12. The left is an LGSVL simulation istance in Borregas Ave with random traffic, and the right shows the UI monitor of Apollo, namely Dreamview.

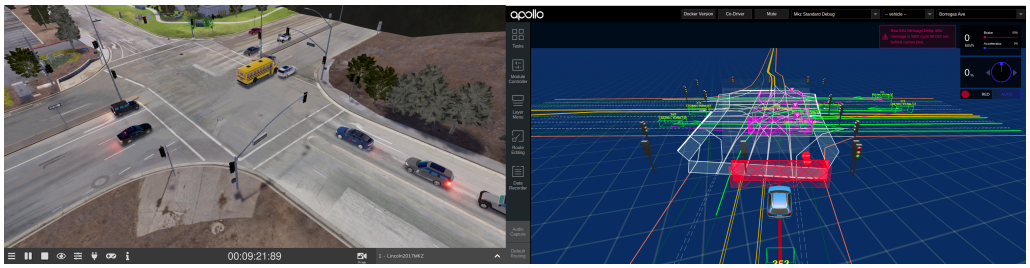

Figure 3.12 Apollo in LGSVL

The autonomous cars have two essential key parameter indicators (KPI) when it comes to crash analysis. The two KPIs are:

- Time to Collision (TTC) (Archer, 2005)
- Distance to Collision (DTC)

DTC is a sub-part of TTC and TTC is known to be objectively good at analyzing car crashes, in the sense that it also includes DTC and other parameters. Thus, the tests are generated by using TTC as the KPI.

After the crash time is calculated with the afore-mentioned strategy. Initial TTC can be calculated by simply taking the difference between current simulation time and crash time. The tests are conducted such that control of the ego car is overridden by the calculated speed and position from the analysis step until a user or computer defined time to collision (TTC) or distance to collision parameter is satisfied, where the controls are given to the autonomous driving stack. By default the TTC value is selected as a list of values in increments of 100 milliseconds starting from the maximum value possible (start of the video $t = 0$) and 1 second to crash.

4. User and Case Studies

To evaluate our approach of video analysis and scene reconstruction we have conducted a user study among a collection of heterogeneous age and jobs of participants. Along with the user study we have also conducted a set of experiments to evaluate performance of Apollo AD stack on selected accidents, which would show the capabilities of our test case generation part of the pipeline. Evaluating car crashes can be hard and subjective to people's perspective. One of the main reasons for us to evaluate our simulation performance with a user study is the subjectivity, caused by the nature of car accidents. Another important reason was that the lack of ground truth and labeled data to evaluate our study. A similar study that is introduced prior in this thesis (Huynh et al., 2019), also uses a similar approach to evaluate simulation performance.

4.1 Subjects

Our evaluation consists six different videos that are taken from various YouTube videos. These videos are particularly selected to evaluate different aspects of a car accident. Table 4.1 show the selected aspect of each accident.

Table 4.1 Aspects of the accidents that are used in evaluation

Video	**Crash Aspect**
1	Side Impact, Footage from Front
2	Side Impact, Footage From Back
3	Front Impact
4	Side Impact, Tilting Car
5	Rear Impact
6	Dragging, Night Accident

4.2 Evaluation of the User Study

In the user study afore-mentioned six videos have been shown to participants. The participants were given a short introduction on the thesis motivation, videos and survey questions, then asked to first answer personal questions, then answer each question after seeing the respective video. The participants were allowed to watch the videos multiple times. The videos here are only to evaluate the performance of scene reconstruction and analysis steps and do not contain any autonomous car. A sample image from the video can be seen in Figure 4.1. Each video here is a collage of 4 videos:

- Top left consists the original raw footage of the car crash.
- Top right, consists the output of the image processing pipeline that tracks the cars, assigns a unique ID for each car and estimates the speed of the cars in km/h.
- Bottom left, consists the bird eye view of the scene in 2D with the unique ID's of the cars (which are represented as arrows)
- Bottom right, consists the final 3D reconstruction of the same car crash in the physics simulation, LGSVL.

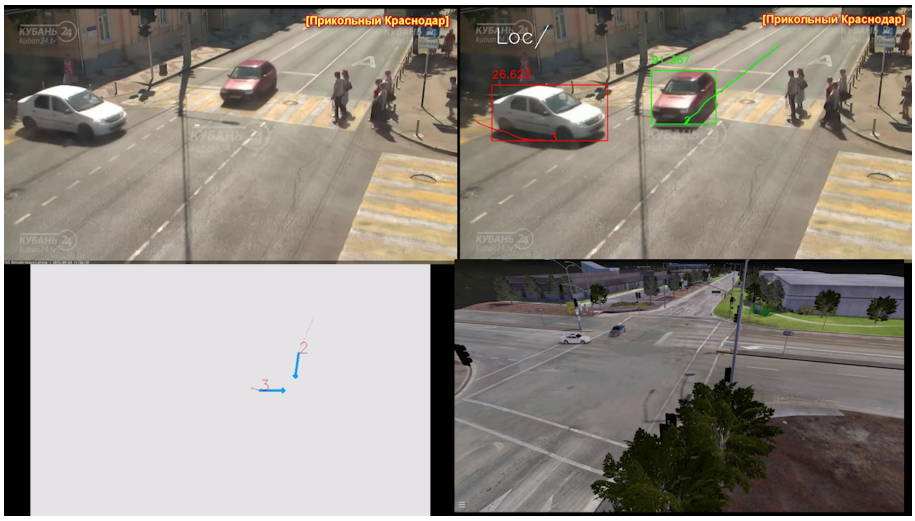

Figure 4.1 A sample frame that is taken from the Video 4 that is shown to user study participants

The user study consists of the following personal questions to evaluate how heterogeneous the test group is:

- What is your current job?: Short text answer
- What is your education level?: Multiple choice between High School, Undergraduate, Master, PhD and Other
- What is your age?: Multiple choice between 15-19, 20-24, 25-29, 30-34, 35-44, 45-54 and 55+

And the following question were repeated for each video:

- What was the cause of the crash?: Drivers, Vehicles Environment, Not Sure, Other

The following statements were also repeated and asked to be graded within a likert scale that contains, Strongly Disagree, Disagree, Neutral, Agree and Strongly Agree:

- All the cars directly involved in the accident are detected. (Note that the cars that are not directly involved in the accident are not required to be detected.) (Top right video)
- Motions of the cars look realistic. (Bottom right video)
- Speeds of the cars look realistic. (Top right video)
- Simulation looks realistic. (Note that the environment, including the streets, traffic lights, pedestrians, buildings, and weather conditions are not meant to be simulated. Answer this question by solely focusing on the cars motions.) (Bottom right video)
- Simulation includes the cause of the crash (Bottom right video)

4.3 Analysis and Discussion of User Study

The participant were a heterogeneous group of 37 people with a variety of different jobs, changing from field experts to students, with ages dominant around 20 to 29. Although the dominant education level was undergraduate, our study included responses from a mix of master's and PhD levels of participants as well. Figures 4.2, 4.3 and table 4.2 shows the respective answers to personal questions.

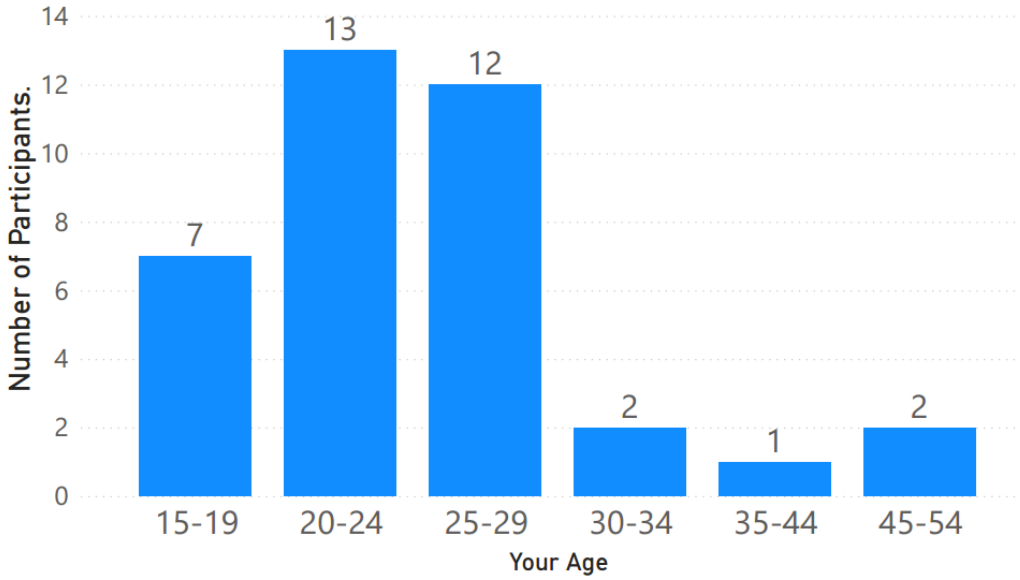

Figure 4.2 Age distribution among user study participants

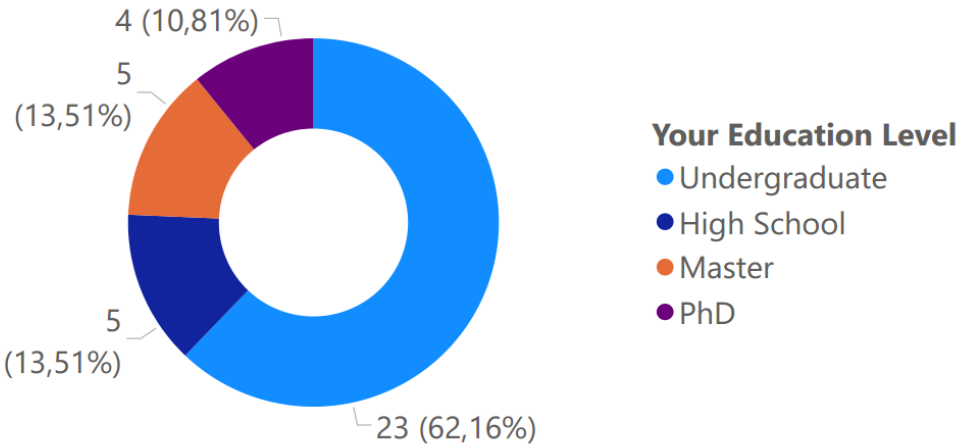

Figure 4.3 Level of education distribution among user study participants

Table 4.2 Job distribution among user study participants

Job	**Number of Participants**
Student	17
Software Engineer	5
Research Assistant	3
Control Engineer	3
Accountant	1
Brand Manager	1
Chief of Staff	1
Human Resources	1
Naval Architecture And Marine Engineer	1
Researcher	1
Teacher	1
Worker	1

One of our studies top priority is to include the cause of the crash in the simulated environment, which would create the backbone of a realistic test, that can be used to show capabilities of AD stacks in real life accident scenarios. Thus, the first question asked to the participants was "What was the cause of the crash?". Figure 4.4 shows the overall results that is found by taking the average response of each video, where users could select from Drivers, Environment, Vehicles, Not Sure and Other. These causes were determined from an official NHTSA report (Thorn et al., 2018). Table 4.3 shows individual results for each video. These answers imply that most accidents are considered to be driver fault, and can be avoided with an AD stack. Although there is an overall agreement on driver fault in each video, these responses show that there is much subjectivity between each participant.

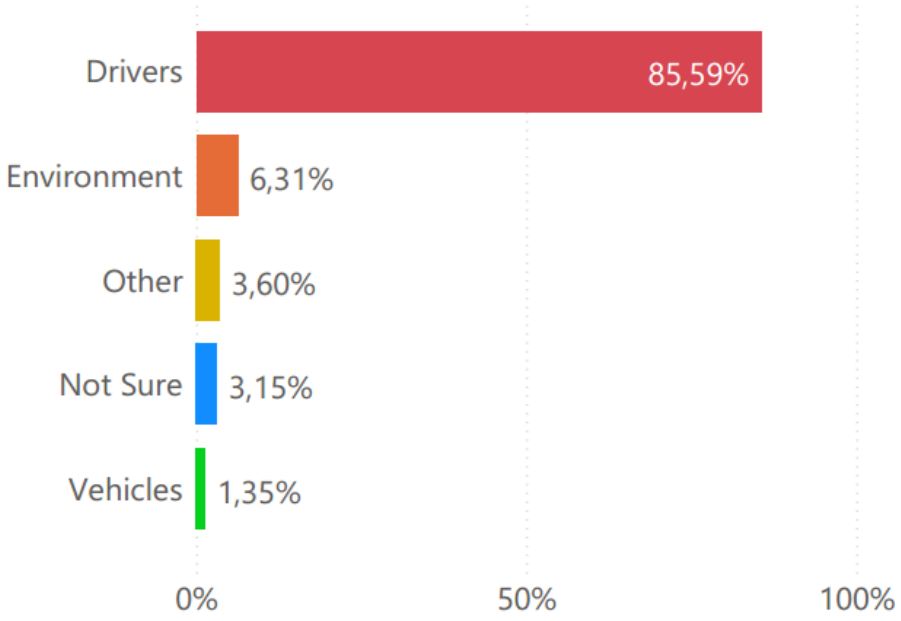

Figure 4.4 Overall responses to cause of the crash of different videos

Table 4.3 % of participants response to cause of the crash in video subjects

Cause of the crash / % of Participants	**Drivers**	**Environment**	**Vehicles**	**Unsure**	**Other**
Video 1	**94.59**	2.70	0	2.70	0
Video 2	**62.16**	13.51	13.51	8.11	2.70
Video 3	**75.68**	10.81	5.41	5.41	2.70
Video 4	**97.30**	0	0	2.70	0
Video 5	**89.19**	8.11	0	2.70	0
Video 6	**94.59**	2.70	0	0	2.70

The first question that is graded in the likert scale was to analyze our image processing pipelines performance and participants were asked to answer this question solely on the cars that are directly involved in the accident. Figure 4.5 shows the overall response to this question, while Table 4.4 shows answers to individual answers to this question. From the responses we can confidently say that our image processing pipeline was successful on detecting and tracking vehicles that are involved in the accident.

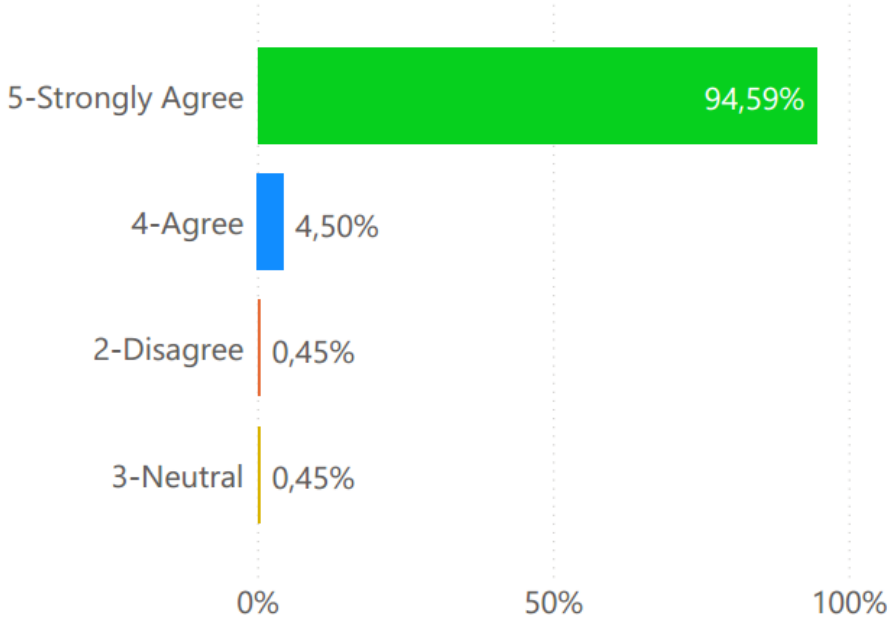

Figure 4.5 Overall responses to detection success of different videos

Table 4.4 % of participants response to detection success in video subjects

Detection Success / % of Participants	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Video 1	91.89	8.11	0	0	0
Video 2	97.3	2.7	0	0	0
Video 3	94.59	2.7	0	2.7	0
Video 4	89.19	10.81	0	0	0
Video 5	94.59	2.7	2.7	0	0
Video 6	100	0	0	0	0

The second question was asked for the motions that are conducted in the physics simulator were realistic or not. This question was asked to evaluate our rotation, translation and trajectory estimation of the vehicles in the accident video. Figure 4.6, which is showing the overall response among all videos, show that our average is on agree. This implies that our rotation, translation and trajectory estimations were not as successful as our detection algorithm but still accurate enough to convince different participants sense of realism. Table 4.5 shows individual results for each video, where we either agree or high-end neutral in each individual video.

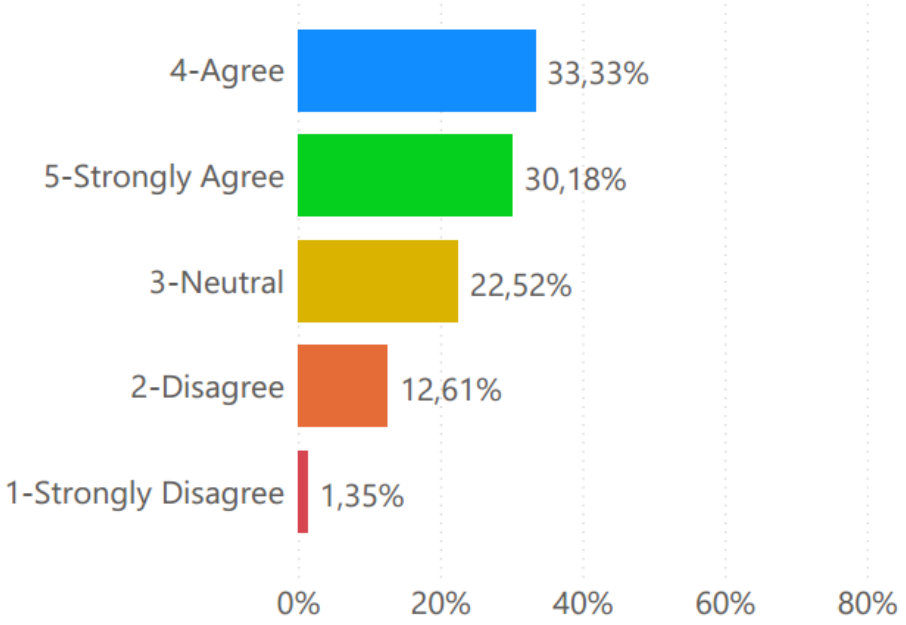

Figure 4.6 Overall responses to motion success of different videos

Table 4.5 % of participants response to realistic motions in simulation in video subjects

Realistic Motions / % of Participants	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Video 1	18.92	59.46	13.51	8.11	0
Video 2	13.51	21.62	43.24	18.92	2.7
Video 3	45.95	32.43	10.81	10.81	0
Video 4	43.24	21.62	24.32	8.11	2.7
Video 5	43.24	24.32	18.92	13.51	0
Video 6	29.73	27.03	24.32	16.22	2.7

The third question asks if the simulation includes the cause of the crash, which is a subjective but crucial aspect of our studies success. This question can also be interpreted as "Does the simulation contain the elements that lead to an accident?". Figure 4.7 shows the overall responses averaged from each video, and Table 4.6 shows individual answers for each video. The answers indicate that our study includes the cause of the crash in every video by averaging strongly agree. This shows that our simulation is realistic in the sense that if an autonomous car passes the test in simulation, it is believed that it would be able to avoid accident in real life too.

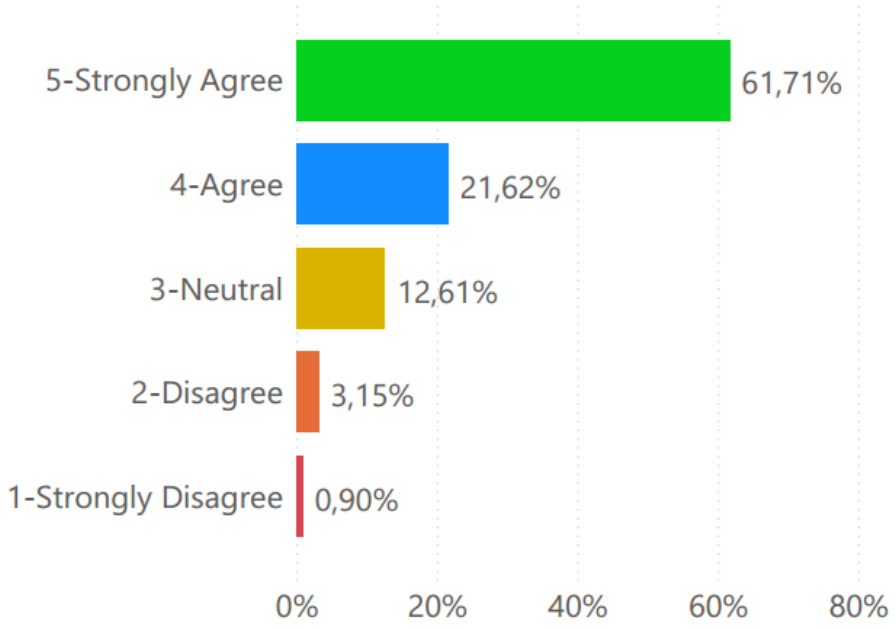

Figure 4.7 Overall responses to including cause of the crash in simulation of different videos

Table 4.6 % of participants response to inclusion of the crash in simulation in video subjects

Simulation Includes Cause of Crash / % of Participants	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Video 1	62.16	24.32	13.51	0	0
Video 2	45.95	29.73	21.62	2.7	0
Video 3	67.57	21.62	5.41	5.41	0
Video 4	64.86	21.62	13.51	0	0
Video 5	67.57	10.81	10.81	5.41	5.41
Video 6	62.16	21.62	10.81	5.41	0

The fourth question is a general evaluation question that is asked to participants on how realistic in general they find the simulation. This question involves the environment, crash, physics, rendering and every other factor that can affect the simulation's realism aspect. This question partially includes our selection of simulation along with our scene reconstruction algorithm. Table 4.7 shows % number of participants that responded that particular answer in each video. Figure 4.8 gives an overall look, averaging all videos. From the responses we can interpret that our simulation is realistic and our selection of simulation is well fit, but there is much room for improvement as we averaged high-end agree overall and between neutral, agree and strongly agree on videos. Particularly the simulation of the second video

was not found realistic, caused by a drift in one of the vehicle's rotation at the start of the video.

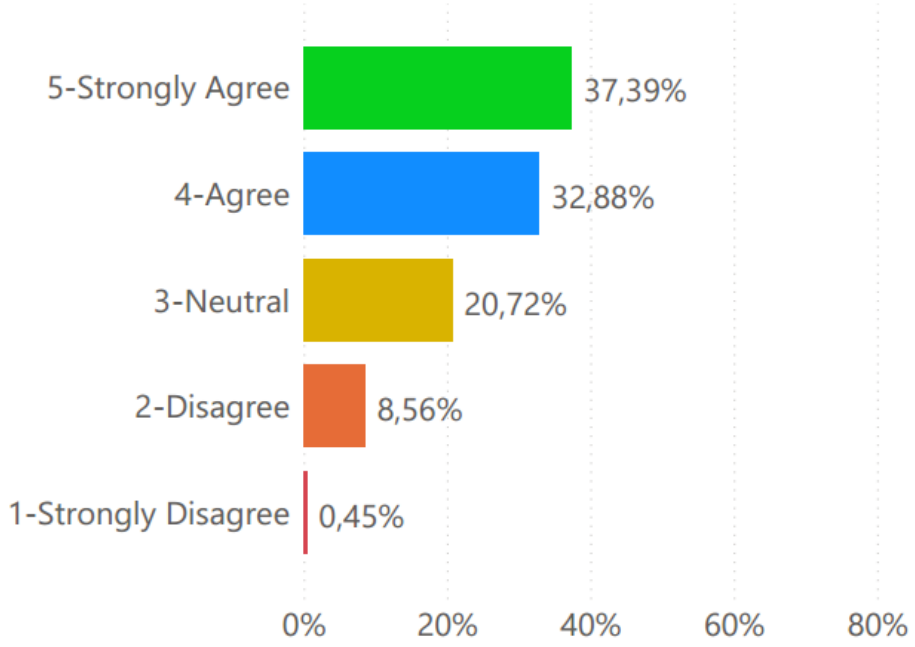

Figure 4.8 Overall responses to realistic simulations of different videos

Table 4.7 % of participants response to realistic simulations in video subjects

Realistic Simulation / % of Participants	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Video 1	43.24	51.35	2.7	2.7	0
Video 2	24.32	29.73	35.14	10.81	0
Video 3	35.14	40.54	18.92	5.41	0
Video 4	37.84	24.32	21.62	13.51	2.7
Video 5	48.65	24.32	18.92	8.11	0
Video 6	35.14	27.03	27.03	10.81	0

The final question of the user study was whether the speeds of the cars look realistic or not. Although this could have been measured with a ground truth data, there is no dataset that includes car accidents with speeds labeled, and the information that is measured by the local police through radars and other sensors are confidential and not available on request. We have asked users to answer this question through how fast they think the vehicles are moving in the video in km/h. The overall answer is strongly agree as can be seen from Figure 4.9 and Table 4.8 shows that we average agree or strongly agree on every video subject. The results show that our speed estimation estimates speeds realistically according to the participants opinion.

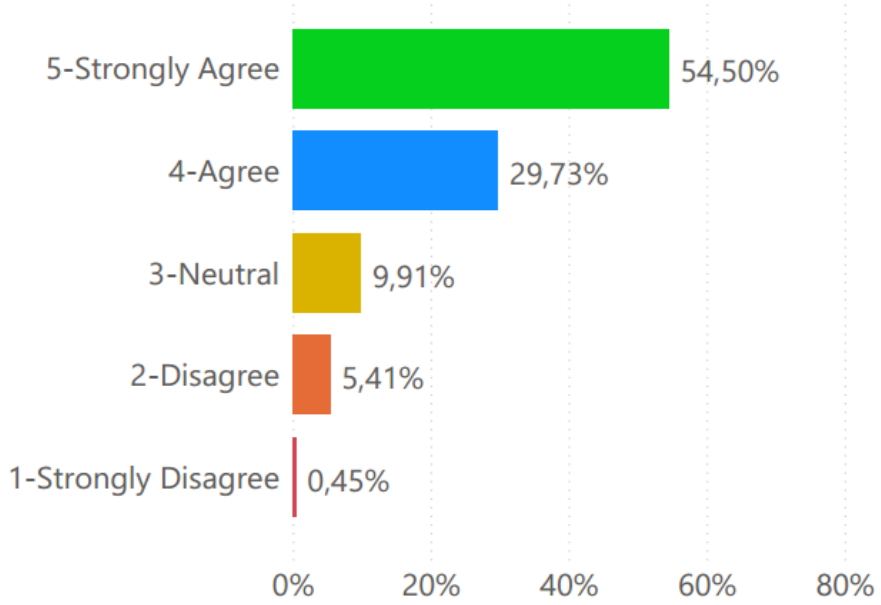

Figure 4.9 Overall responses to realistic speeds of different videos

Table 4.8 % of participants response to realistic speeds in video subjects

Realistic Speed / % of Participants	**Strongly Agree**	**Agree**	**Neutral**	**Disagree**	**Strongly Disagree**
Video 1	48.65	37.84	10.81	2.7	0
Video 2	32.43	43.24	18.92	2.7	2.7
Video 3	67.57	24.32	0	8.11	0
Video 4	72.97	21.62	5.41	0	0
Video 5	59.46	18.92	13.51	8.11	0
Video 6	45.95	32.43	10.81	10.81	0

In the light of our user study, we can interpret that our simulation serves as a solid proof of concept, that includes realistic speed estimation, tracking information and realistic simulation. The accident is convincingly recreated in the physics simulator according to most participants. Although there is still much room for improvement and considering the various limitations of this study, even the current version of this study serves as a solid foundation for future works.

4.4 Case Study

Along with the user study, we've tested Apollo 5.0 AD stack by utilizing the test case generation steps in addition to analysis and scene reconstruction, to produce automated test cases and run the tests over the same six videos to evaluate the performance of Apollo 5.0 on avoiding these accidents. All the tests are conducted in a computer with following configuration:

- CPU: i7-9750H
- RAM: 16GB
- GPU: Nvidia GeForce GTX 1660Ti
- OS: Ubuntu 18.04.5
- Kernel: 5.4.0-74

Other versioning of software that is used in this case study are as follows:

- LGSVL: 2021.1
- LGSVL API: Modified version of Preview (Alpha) Release
- Apollo: 5.0
- Python: 3.6.9

The test process were conducted such that a specific vehicle was selected manually for each video and the results were analyzed in terms of a crash happening or not. The only variable that is changed between tests are time to collision with increments of 0.1 seconds until time to collision is evaluated as less than 1 second. Tables 4.9, 4.10, 4.11, 4.12, 4.13, 4.14 shows the results for each respective crash that is derived from afore-mentioned videos.

Table 4.9 Accident results of Apollo 5.0 on the first crash

Video	**Time To Collision**	**Tested Car ID**	**Crash**
1	4.0	3	No
1	3.9	3	No
1	3.8	3	No
1	3.7	3	No
1	3.6	3	No
1	3.5	3	No
1	3.4	3	No
1	3.3	3	No
1	3.2	3	No
1	3.1	3	No
1	3.0	3	No
1	2.9	3	No
1	2.8	3	No
1	2.7	3	No
1	2.6	3	No
1	2.5	3	No
1	2.4	3	No
1	2.3	3	No
1	2.2	3	No
1	2.1	3	Yes
1	2.0	3	Yes
1	1.9	3	Yes
1	1.8	3	Yes
1	1.7	3	Yes
1	1.6	3	Yes
1	1.5	3	Yes
1	1.4	3	Yes
1	1.3	3	Yes
1	1.2	3	Yes
1	1.1	3	Yes

Table 4.10 Accident results of Apollo 5.0 on the second crash

Video	**Time To Collision**	**Tested Car ID**	**Crash**
2	1.8	8	Yes
2	1.7	8	Yes
2	1.6	8	Yes
2	1.5	8	Yes
2	1.4	8	Yes
2	1.3	8	Yes
2	1.2	8	Yes
2	1.1	8	Yes

Table 4.11 Accident results of Apollo 5.0 on the third crash

Video	**Time To Collision**	**Tested Car ID**	**Crash**
3	3.6	2	No
3	3.5	2	No
3	3.4	2	No
3	3.3	2	No
3	3.2	2	No
3	3.1	2	No
3	3.0	2	No
3	2.9	2	No
3	2.8	2	No
3	2.7	2	No
3	2.6	2	No
3	2.5	2	No
3	2.4	2	No
3	2.3	2	No
3	2.2	2	No
3	2.1	2	No
3	2.0	2	No
3	1.9	2	No
3	1.8	2	No
3	1.7	2	No
3	1.6	2	Yes
3	1.5	2	Yes
3	1.4	2	Yes
3	1.3	2	Yes
3	1.2	2	Yes
3	1.1	2	Yes

Table 4.12 Accident results of Apollo 5.0 on the fourth crash

Video	**Time To Collision**	**Tested Car ID**	**Crash**
4	1.5	2	Yes
4	1.4	2	Yes
4	1.3	2	Yes
4	1.2	2	Yes
4	1.1	2	Yes

Table 4.13 Accident results of Apollo 5.0 on the fifth crash

Video	**Time To Collision**	**Tested Car ID**	**Crash**
5	2.1	4	No
5	2.0	4	No
5	1.9	4	No
5	1.8	4	Yes
5	1.7	4	Yes
5	1.6	4	Yes
5	1.5	4	Yes
5	1.4	4	Yes
5	1.3	4	Yes
5	1.2	4	Yes
5	1.1	4	Yes

Table 4.14 Accident results of Apollo 5.0 on the sixth crash

Video	**Time To Collision**	**Tested Car ID**	**Crash**
6	3.6	2	No
6	3.5	2	No
6	3.4	2	No
6	3.3	2	No
6	3.2	2	No
6	3.1	2	No
6	3.0	2	No
6	2.9	2	No
6	2.8	2	No
6	2.7	2	No
6	2.6	2	No
6	2.5	2	No
6	2.4	2	No
6	2.3	2	No
6	2.2	2	No
6	2.1	2	No
6	2.0	2	No
6	1.9	2	No
6	1.8	2	No
6	1.7	2	No
6	1.6	2	No
6	1.5	2	No
6	1.4	2	No
6	1.3	2	No
6	1.2	2	No
6	1.1	2	No

As can be analyzed from these test results, every accident that we have conducted the tests on has a “point of no return” as expected. The point of no return is defined as the AD stack under test, cannot prevent the accident after this point. This can either be between start of the appearance of the ego car and 1 second, below 1 second or prior to appearance of the car in the video, or after there is 1 second to collision. For example the first video has the ”point of no return“ in between, the sixth video has it under 1 second mark and the fourth video has it before the ego car appeared in the video. The point of no return being before the accident implies that we need to analyze back in time of the video to find a non-accident test run,

which our study currently is not capable of. If the point of no return is less than 1 second the tests can be rerun again to manually include times under 1 second TTC.

This tests show that AD stacks can avoid the accident when they are replaced with human drivers in certain cases. This also proves the capability of our software to perform analysis, scene reconstruction and test case generation steps successively to achieve a full end to end pipeline to generate runnable test cases for AD stacks using a physics simulator with the only input resource being the accident videos.

The time our software takes for each step from image processing pipeline to test case generation and execution, using a 2D video input from scratch with our current configuration is given in Table 4.15. Here we can see a strong correlation between number of test cases generated and test case execution time, which has the most significant percentage of total time in an end-to-end run. Test case generation is considered only generating runnable test case files, which happens quickly, using the apriori information from scene reconstruction step.

Table 4.15 Time needed to execute each step in our study

Video	**Video Length**	**Analysis**	**Scene Rec.**	**Test Case Gen.**	**No. of Test Cases**	**Test Case Exe.**	**Total Time**
1	10	300	130	13	30	2250	2693
2	5.8	174	75	8	8	566.4	823.4
3	6.3	189	82	9	26	1853.8	2133.8
4	3.2	96	42	6	5	341	485
5	4	120	52	7	11	759	938
6	6.2	186	81	9	26	1851.2	2127.2

5. Threats to Validity

This study contains a variety of restrictions and assumptions causing threats to validity of the experiments. These threats can be analyzed for each step in our study.

Much of the restrictions that affects the other steps are stemming from the analysis step. Here we use a multi process image processing pipeline to achieve a detailed analysis of how the crash has happened. Various different neural networks that we use, to which comes with various assumptions and accuracy limitations that are stated in chapter 3. Specifically the speed estimation process has various different assumptions including maximum and minimum speeds and average speeds of the cars. Camera calibration process assumes that there is at least one traffic line where at least one vehicle is aligned with as well as assuming every car analyzed are an average sedan size car. Although we analyze the roads in object segmentation step we don't use it and don't analyze what type of road it is along with its material. There is no analysis on weather is done and wetness of the roads. Although all these assumptions and limitations can not be ignored, our study still represents the car crashes in a realistic manner according to our user study. However, the user study represents a selection of limited number of crashes with thousands of different crash videos remain untested, creating a potential threat the scope of our user study.

In the scene reconstruction step, we use only one simulator, LGSVL, however the capabilities of this study can theoretically extends beyond one simulator but it won't be a safe assumption unless this hypothesis is tested with different simulators. Using only one simulator also comes with limiting our capabilities to this simulator and making the same assumptions the simulator makes. Another and potentially the biggest threat to validity in this step is using the same map for every accident without actually constructing a map. Every video footage that we use in our experiments are in different locations with different environmental conditions and varying types of road and road material. There is a great number of factors that can contribute to a car accident, many being related to environment. However our study can only consider a small number of these factors in its current state without our potential

future improvements stated in chapter 7 of this thesis. The simulator also contains a certain amount of different vehicles to be tested, however each brand and each model cars are different than each other which can be a significant factor contributing to an accident happen.

In the test case generation step, similar to scene reconstruction step we only consider one autonomous driving driving stack, with only one ego car option. Again, although our software is constructed in a modular manner to allow any autonomous driving using communication bridges, this is not tested. Test are generated in limited number of configurable parameters which can only represent a part of the many factors that contributes to a car accident.

In the user study, we do not ask the difference between simulation and original car crash videos, and instead we give them the information on which is the simulated video and which is the original crash. This may have caused users leaning towards giving more positive answers while they are evaluating how realistic the simulation is.

Although, we have compiled numerous potential threats to validity here, we strongly believe, in parallel with our user study results, that this study serves as a proof of concept and leaves room to improve this to be a potentially widely used way to generate test cases for autonomous cars.

6. Conclusion

We have proposed a method for potentially generating a great number of realistic test cases for autonomous driving stacks using accident footage. We strongly believe that increasing the number of realistic test cases will increase the safety and reliability of autonomous vehicles and increase these developers ability to test their capabilities.

Our method, introduces an offline image processing pipeline that can run a number of analysis to determine each vehicles 3D trajectory, a new novel concept to 3D camera calibration from 2D video footages and improvements on a 3D tracking neural network, CenterTrack. The image processing pipeline is complimented with a scene reconstruction and test case generation steps to complete an end to end test case generation software from a 2D video.

Although this research shows the potential of generating these test cases, there were numerous assumptions that prevented our software from being more realistic. There are quite a few factors that contributes to an accident and our research shows a limited scope of these variables.

We have conducted a user study to understand how realistic our simulations are, and how likely it represents the actual accident. The results of our user study, that is conducted in a variety of range of ages and professions, shows that even though our software has assumptions and limitation on its own, the resultant scenes that are reconstructed in an automated manner were realistic enough and represents the actual accident.

7. Future Work

This study works as a proof of concept, showing that automated car crash tests can be generated by only using 2D video footage without any prior information. However, there is much room to improve in this concept. As stated in chapter 5 we currently have a limited scope of videos that can be analyzed with some strict restrictions and assumptions.

As can be seen from the references this study uses very recently published work. More than 50% of references are published in 2019 and later. There are various limitations of the tools we are using such as LGSVL simulator and various neural networks. In the near future, with the advancements in artificial intelligence and autonomous car technologies the networks we use will become precise and the simulators will have much more realistic physics with improved APIs. We will continue to adapt to these changes also contributing them with open source issues and pull requests and test our software in different simulators to prove our analysis steps flexibility.

Many assumptions of this study can be removed by adding additional processes and further post-processing in the analysis step. We plan to add these processes in near future and also extend our capabilities to automatically creating a map in the analysis step and use it while reconstructing the scenes.

Another important feature we are planning to add is to analyzing the past trajectories of the cars from the video. This is possible since we know the cars trajectory and speed as soon as it appears in the video. Analyzing past trajectories will help to make the tests more realistic.

A final and more trivial future work we are planning to add is to conduct the same (or improved) tests with other autonomous driving stacks, proving the modularity of our program further.

BIBLIOGRAPHY

Archer, J. (2005). Indicators for traffic safety assessment and prediction and their application in micro-simulation modelling : a study of urban and suburban intersections.

ASAM (2020a). OpenDRIVE.

ASAM (2020b). OpenSCENARIO.

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. *Proceedings - International Conference on Image Processing, ICIP*, *2016-Augus*, 3464–3468.

Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv*.

Bulo, S. R., Porzi, L., & Kontschieder, P. (2018). In-place Activated BatchNorm for Memory-Optimized Training of DNNs. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 5639–5647.

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., & Beijbom, O. (2020). Nuscenes: A multimodal dataset for autonomous driving. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (March), 11618–11628.

Caldeira, J., Fout, A., Kesari, A., Sefala, R., Walsh, J., Dupre, K., Khaefi, M. R., Setiaji, Hodge, G., Pramestri, Z. A., & Imtiyazi, M. A. (2020). Improving traffic safety through video analysis in Jakarta, Indonesia. *Advances in Intelligent Systems and Computing*, *1038*(February), 642–649.

Chaabane, M., Zhang, P., Beveridge, J. R., & O'Hara, S. (2021). DEFT: Detection Embeddings for Tracking.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). The Cityscapes Dataset for Semantic Urban Scene Understanding. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 3213–3223.

Coumans, E. (2015). Bullet physics simulation. (pp.1̃).

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T. A., & Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. *CoRR*, *abs/1702.04405*.

Dendorfer, P., Osep, A., Milan, A., Schindler, K., Cremers, D., Reid, I., Roth, S., & Leal-Taixé, L. (2021). MOTChallenge: A Benchmark for Single-Camera Multiple Target Tracking. *International Journal of Computer Vision*, *129*(4), 845–881.

Dosovitskiy, A., Ros, G., Codevilla, F., & Antonio, L. (2017). CARLA : An Open Urban Driving Simulator. (CoRL), 1–16.

Duan, K., Bai, S., Xie, L., Qi, H., Huang, Q., & Tian, Q. (2019). CenterNet: Keypoint triplets for object detection. *Proceedings of the IEEE International Conference on Computer Vision*, *2019-Octob*, 6568–6577.

Games, E. Unreal engine.

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics:

The KITTI dataset. The International Journal of Robotics Research. *The International Journal of Robotics Research*, (October), 1–6.

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 1440–1448.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 580–587.

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2020). Mask R-CNN. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *42*(2), 386–397.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 770–778.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv*.

Huynh, T., Gambi, A., & Fraser, G. (2019). AC3R: Automatically reconstructing car crashes from police reports. *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion, ICSE-Companion 2019*, (May), 31–34.

Kam, H., Lee, S.-H., Park, T., & Kim, C.-H. (2015). Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, *60*, 1–9.

Kato, S., Tokunaga, S., Maruyama, Y., Maeda, S., Hirabayashi, M., Kitsukawa, Y., Monrroy, A., Ando, T., Fujii, Y., & Azumi, T. (2018). Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems. *Proceedings - 9th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS 2018*, (April), 287–296.

Koenig, N. & Howard, A. (2004). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, (pp. 2149–2154 vol.3).

Kumar, A., Khorramshahi, P., Lin, W. A., Dhar, P., Chen, J. C., & Chellappa, R. (2018). A semi-automatic 2D solution for vehicle speed estimation from monocular videos. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, volume 2018-June, (pp. 137–144).

Leal-Taixé, L., Milan, A., Reid, I., Roth, S., & Schindler, K. (2015). MOTChallenge 2015: Towards a benchmark for multi-target tracking. *arXiv:1504.01942 [cs]*. arXiv: 1504.01942.

Lee, J., X. Grey, M., Ha, S., Kunz, T., Jain, S., Ye, Y., S. Srinivasa, S., Stilman, M., & Karen Liu, C. (2018). DART: Dynamic Animation and Robotics Toolkit. *The Journal of Open Source Software*, *3*(22), 500.

Lee, K. H., Lee, Y. J., & Hwang, J. N. (2013). Multiple-kernel based vehicle tracking using 3-D deformable model and license plate self-similarity. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 1793–1797.

Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common objects in context. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial*

Intelligence and Lecture Notes in Bioinformatics), *8693 LNCS*(PART 5), 740–755.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9905 LNCS*, 21–37.

Long, J., Shelhamer, E., & Darrell, T. (2014). Fully convolutional networks for semantic segmentation. *CoRR*, *abs/1411.4038*.

Merkle, F. H. & Discher, C. A. (1964). Controlled-Potential Coulometric Analysis of N-Substituted Phenothiazine Derivatives. *Analytical Chemistry*, *36*(8), 1639–1643.

Milan, A., Leal-Taixe, L., Reid, I., Roth, S., & Schindler, K. (2016). MOT16: A Benchmark for Multi-Object Tracking, 1–12.

Naphade, M., Chang, M. C., Sharma, A., Anastasiu, D. C., Jagarlamudi, V., Chakraborty, P., Huang, T., Wang, S., Liu, M. Y., Chellappa, R., Hwang, J. N., & Lyu, S. (2018). The 2018 NVIDIA AI city challenge. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, *2018-June*, 53–60.

Neuhold, G., Ollmann, T., Bulo, S. R., & Kontschieder, P. (2017). The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes. *Proceedings of the IEEE International Conference on Computer Vision*, *2017-Octob*, 5000–5009.

Piazzi, A. & Guarino Lo Bianco, C. (2002). Quintic G/sup 2/-splines for trajectory planning of autonomous vehicles. (September 2014), 198–203.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 779–788.

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149.

Rong, G., Shin, B. H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., Agafonov, E., Kim, T. H., Sterner, E., Ushiroda, K., Reyes, M., Zelenkovsky, D., & Kim, S. (2020). LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. *2020 IEEE 23rd International Conference on Intelligent Transportation Systems, ITSC 2020*.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 4510–4520.

Smith, R. et al. (2005). Open dynamics engine.

Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., Zhang, Y., Shlens, J., Chen, Z., & Anguelov, D. (2020). Scalability in perception for autonomous driving: Waymo open dataset. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2443–2451.

Tang, Z., Wang, G., Xiao, H., Zheng, A., & Hwang, J.-N. (2018a). Single-camera and inter-camera vehicle tracking and 3d speed estimation based on fusion

of visual and semantic features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, (pp. 108–115).

Tang, Z., Wang, G., Xiao, H., Zheng, A., & Hwang, J. N. (2018b). Single-camera and inter-camera vehicle tracking and 3d speed estimation based on fusion of visual and semantic features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, volume 2018-June, (pp. 108–115).

Technologies, U.

Thorn, E., Kimmel, S., & Chaka, M. (2018). A Framework for Automated Driving System Testable Cases and Scenarios. *Dot Hs 812 623*, (September), 180.

Weng, X., Wang, J., Held, D., & Kitani, K. (2020). AB3DMOT: A baseline for 3D Multi-object tracking and new evaluation metrics. *arXiv*, *1*, 1–4.

Xu, K., Xiao, X., Miao, J., & Luo, Q. (2020). Data Driven Prediction Architecture for Autonomous Driving and its Application on Apollo Platform. In *IEEE Intelligent Vehicles Symposium, Proceedings*, (pp. 175–181).

Yin, T., Zhou, X., & Krähenbühl, P. (2020). Center-based 3D object detection and tracking.

Zhou, X., Koltun, V., & Krähenbühl, P. (2020). Tracking Objects as Points. In *Computer Vision – ECCV 2020*, volume 12349 LNCS, (pp. 474–490).

Zhou, X., Wang, D., & Krähenbühl, P. (2019). Objects as Points.

Zhu, F., Ma, L., Xu, X., Guo, D., Cui, X., & Kong, Q. Baidu Apollo Auto-Calibration System - An Industry-Level Data-Driven and Learning based Vehicle Longitude Dynamic Calibrating Algorithm.