Traffic Accident Reconstruction Using Monocular Dashcam Videos

by

Aniruddh Vinay Chandratre

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2022 by the
Graduate Supervisory Committee:

Georgios Fainekos, Co-Chair
Hani Ben Amor, Co-Chair
Giulia Pedrielli

ARIZONA STATE UNIVERSITY

December 2022

ABSTRACT

Automated driving systems (ADS) have come a long way since their inception. It is clear that these systems rely heavily on stochastic deep learning techniques for perception, planning and prediction, as it is impossible to construct every possible driving scenario to generate driving policies. Moreover, these systems need to be trained and validated extensively on typical and abnormal driving situations before they can be trusted with a human life. However, most publicly available driving datasets only consist of typical driving behaviors. On the other hand, there is a plethora of videos available on the internet that capture abnormal driving scenarios, but they are unusable for ADS training or testing as they lack important information such as camera calibration parameters, and annotated vehicle trajectories.

This thesis proposes a new toolbox, DeepCrashTest-V2, that is capable of reconstructing high quality simulations from monocular dashcam videos found on the internet. The toolbox not only estimates the crucial parameters such as camera calibration, ego motion, surrounding road user trajectories, but also creates a virtual world in Car Learning to Act (CARLA) using data from OpenStreetMaps to simulate the estimated trajectories. The toolbox is open-source and is made available in the form of a python package on GitHub [1].

[1] `https://github.com/C-Aniruddh/deepcrashtest_v2`

To my parents and my sister

ACKNOWLEDGMENTS

I would first like to express my heartfelt gratitude to my committee co-chairs, Dr. Georgios Fainekos and Dr. Heni Ben Amor, and my committee member Dr. Giulia Pedrielli for giving me the opportunity to work on such an exciting idea, for guiding me and for supporting me throughout my master's graduate program.

The work presented in this thesis would not have been possible without the immense support and help of my teammates Tanmay Khandait, Quinn Thibeault, Jacob Anderson, and Tomas Hernandez Acosta. I would like to especially thank Tanmay Khandait and my sister, Yuga Chandratre, for guiding, supporting, and encouraging me through different phases of life.

I would also like to thank Aditya Kalyanaraman, Chinmay Vad, Pragya Malakar, Anthony Crone, and Pratyush Nagare for their friendship and for making my time in Tempe memorable.

Finally, I would like to extend my deep and sincere gratitude to my father, Dr. Vinay Chandratre, and my mother, Mrs. Shirisha Chandratre for their unparalleled love and support.

This work is supported by the DARPA ARCOS program under contract FA8750-20-C-0507.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

INTRODUCTION

Transportation is one of the critical pillars of mobility in society, as it allows the basic access and development needs of individuals, companies, and society to be met safely and in a manner consistent with human and ecosystem health [1]. Among the known transportation modes available in today's world - air, sea, and land, land transportation is the most available and preferred mode of transport [2]. The landscape of land transportation has changed substantially over the last few centuries, especially after the introduction of the Automobile, as it provides a faster, better, and more convenient way of moving from one place to the other [3].

Automobiles have been around for nearly a century. In the early days, an automobile was a luxury only a few could afford. Still, with growing automation, it became increasingly easier for manufacturers to mass produce cars, and in turn, make the automobile a household item [4]. It is estimated that in the US, for every 10 people, there are 7 cars [5]. While it became significantly easier for humans to buy and use cars, it had a downside - with more cars on the road, the probability of being involved in a car crash started trending upward, making automobiles unsafe.

Over the last few decades, there has been an enormous amount of work to make the automobile safer, such as the introduction of passive safety features like crumple zones, airbags, and seat-belts along with active safety features such as Automatic Emergency Braking (AEB), Forward Collision Warning, Adaptive Cruise Control, Lane Departure Warning, Blind Spot monitoring and more. Figure 1.1 represents the trends in traffic casualty in the US from 1913 to 2020. It is clear that with the introduction of passive and active safety features, the rate of a car crash for

every 10,000 vehicles has decreased by 30 folds, even though the absolute number of fatalities has increased over the years.

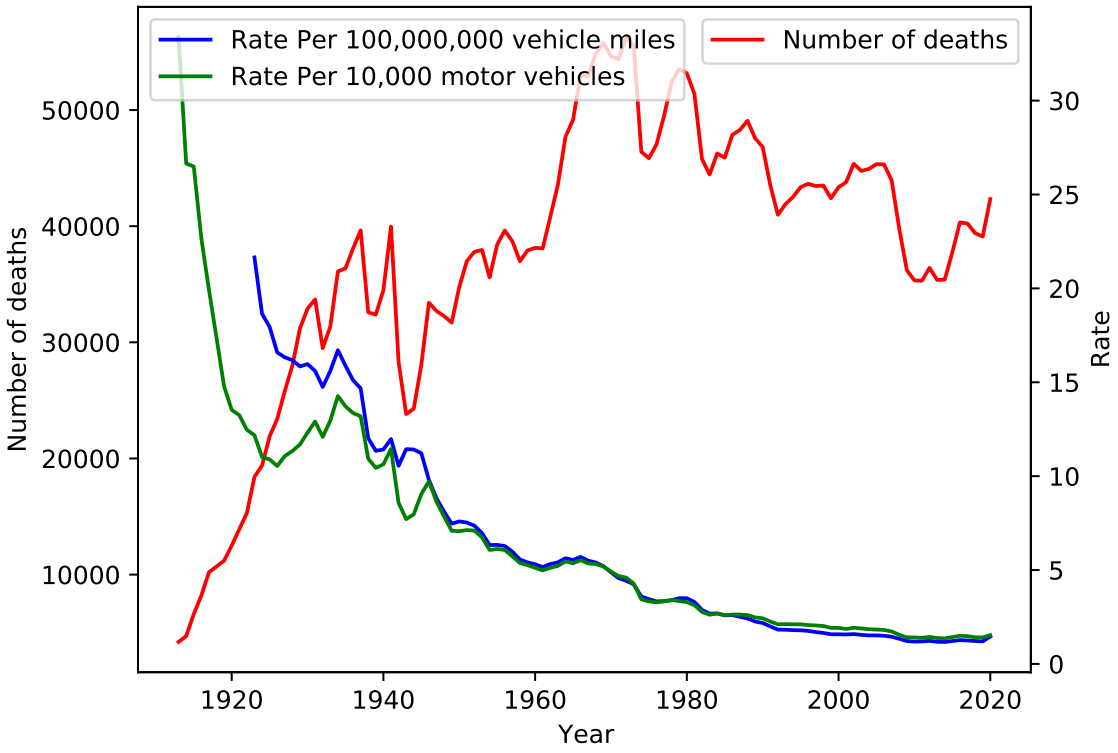


Figure 1.1: Trends in Traffic Casualty in the US (1913–2020). Data from [3]

As active safety technology evolves, the million-dollar question is, where do we go from here?

"Self-driving cars are the natural extension of active safety and obviously something we should do."

-Elon Musk

The race for self-driving cars began back in 1939 when General Motors envisaged that cars can drive themselves in the Futurama exhibit. In the following years, the

domain of automated driving has seen substantial research and developments [6, 7, 8]. The DARPA Grand and Urban Challenge competitions [9] changed the playing field by offering a uniform and modern testing environment for automated cars.

With the availability of computing resources in the AI boom over the last decade, the field of automated driving has seen exponential growth, where we went from the Stanley [10], the 2005 DARPA Grand Challenge winner, which could drive itself 132 miles in a controlled environment to level 4 automated cars being tested publicly on roads, in the span of just 15 years. It is estimated that there are currently nearly 1,400 level 4 cars being tested by 80 different organizations on public US roads [11]. It is also estimated that nearly 92% of currently available new cars have at least level 2 automation capabilities [12]. To precisely classify and evaluate self-driving cars, the Society of Automotive Engineers (SAE) standardized the level of active safety in terms of the need for a human to intervene in the task of driving [13] as summarised in Table 1.1. The standards introduce the term Automated Driving System (ADS) to indicate the active safety controller and also take into consideration the domain in which the ADS will operate in, known as the Operational Design Domain (ODD).

Table 1.1: SAE Levels of Automation (H: Human; A: ADS) [13]

Task	**Level**					
	L0	**L1**	**L2**	**L3**	**L4**	**L5**
Basic control [1]	H	H/A	A	A	A	A
Monitoring environment	H	H	H	A	A	A
Fallback	H	H	H	H	A	A
ODD	Limited	Limited	Limited	Limited	Limited	Unlimited

1.1 Motivation

It is impractical and implausible to model every possible driving scenario and simulate it for creating driving policies that would be safer than a human [14]. For

[1] Basic control includes steering, accelerating and braking.

this reason, Automated Driving Systems heavily rely on stochastic deep learning techniques for perception, planning, and prediction. Deep learning techniques require an enormous amount of data. However, most openly available ADS datasets only provide typical driving data. Due to this, organizations working on ADS have fleets of cars roaming around cities collecting data that is used for training the deep learning models used in the system. The data required by ADS for training often includes a combination of data from a lot of sensors in order to be usable. This creates a limitation that the car should actually be near a traffic incident while it is happening for it to be able to collect any meaningful data in atypical situations. Furthermore, this puts the test driver in a risky situation.

There is a plethora of videos on the Internet that show minor and major crashes, sometimes due to the fault of a driver and sometimes due to road conditions. However, the majority of these videos were captured with a monocular dashcam with unknown camera parameters, rendering them unsuitable for training automated vehicles. The goal of this thesis is to generate high-fidelity simulations using the dashcam videos found on the internet so that we can utilize them to extract valuable collision data and use them to train the ADS. Figure 1.2 demonstrates the end result of the toolbox on a randomly chosen video from YouTube.

1.2 Contribution

This work introduces an end-to-end framework that

1. Extracts temporal ego vehicle motion
2. Extracts 3D temporal trajectories for other road users (vehicles, pedestrians, and cyclists)
3. Generates a 3D map in Car Learning to Act (CARLA) [15] that includes road

networks, nearby buildings, sidewalks, and road objects

4. Replays the scenario in CARLA for further testing and data collection.

The tool introduced in this thesis mainly relies on monocular dashcam videos for all the tasks listed above. Even though the system can work when we do not have the camera calibration parameters or the associated Global Positioning System (GPS) trace, it greatly helps in performance and accuracy when these parameters are made available. The tool is distributed as a standalone python package, and all the Application Programming Interfaces (APIs) are well-documented for ease of use and further development. In order for users to be able to use the tool, the users will need to build CARLA from the source and enable the map generation pipeline, as described in the tool documentation.

1.3 Structure

The manuscript is divided into the following sections:

- Chapter 2 discusses related work and existing methods.
- Chapter 3 describes the problem formulation, constraints, and assumptions
- Chapter 4 describes the trajectory extraction framework including methods used for vehicle detection, tracking, ego-motion estimation, lane detection, and online camera calibration.
- Chapter 5 describes the map generation framework including methods used for extracting information from OpenStreetMaps [16], mesh generation, satellite imagery extraction, and choosing relevant CARLA assets when constructing the world.

- Chapter 6 describes the replay framework where the extracted trajectories are replayed in the constructed world.
- Chapter 7 discusses the qualitative results of the framework
- Chapter 8 concludes the work with ideas for future work.

(a) Frame = 83

(b) Frame = 143

(c) Frame = 180

(d) Frame = 256

Figure 1.2: Side-by-Side Comparison of Input Video Frames From a YouTube Video and the Resulting Simulation at 4 Different Frames. Input Frame on the Left, Resulting Simulation in CARLA on the Right.

Chapter 2

RELATED WORK

There is a baseline established for the task of extracting vehicle trajectories and replaying them in a vehicle simulator in [17]. However, the problem is constrained to scenarios that are short (typically 10-20 seconds), the roads do not contain any intersections, the roads are relatively straight, and the vehicles are not occluded from the ego vehicle point of view at any given time. Additionally, the map has to be created by hand and thus creates a limitation for generating simulations on a large scale.

The proposed solution processes a video in four stages - (1) Estimation of camera calibration parameters (2) Extraction of ego-motion, (3) Detection, tracking, and trajectory extraction of other road users, (4) Procedural map generation. Each of these stages makes use of a combination of state-of-the-art Convolutional Neural Networks (CNNs) and post-processing techniques. The next sections discuss the related works for each of the five processing stages.

2.1 Estimating Camera Calibration

Geometric camera calibration is the process of estimating the parameters of a lens and image sensor of an image or a video. These parameters are used for the correction of lens distortion, estimating the size of an object in world units, or determining the location of a camera in the scene. It is important to correctly estimate camera calibration parameters for accurate scene representation in 3D world units. Zhang et al. [18] introduce a Pinhole camera model along with a flexible technique to easily calibrate a camera. The technique requires the camera to observe the same

planar pattern in at least two different orientations. [19] introduces a new algorithm for pinhole camera calibration. However, these approaches require a planar pattern (generally a chessboard) for calibration and are thus reliant on man-made scenes. This might not be the case in many practical applications.

Yan et al. [20] introduce a large-scale dataset, FocaLens, which is designed for single-image focal length estimation. [20] also proposes a new focal length estimation model, which exploits multi-scale detection architecture to encode object distributions in images to assist focal length estimation. It is then demonstrated that a model trained on FocaLens achieves state-of-the-art results without distinct geometric cues. Workman et al. [21] introduce a novel architecture using a deep CNN, trained on natural images collected from the Internet to directly estimate the focal length by using raw pixel intensities as input features. Fung et al. [22] demonstrated a more simple and straightforward approach that reformulates the perspective camera equations using lane line annotations to estimate camera parameters.

2.2 Ego Motion Estimation

Accurate localization of a vehicle is a fundamental challenge and one of the most important tasks for autonomous driving [23]. Visual Odometry (VO), and Simultaneous Localization and Mapping (SLAM) are robust techniques used for this process. VO and SLAM techniques can be used in signal-denied areas, and thus they are able to overcome the limitations imposed by GNSS-based techniques, whilst maintaining a superior accuracy [24]. The process of estimating an agent's ego-motion using only the input of a single or multiple cameras attached to it is known as VO. SLAM, on the other hand, is a process in which a robot is required to localize itself in an unknown environment while also building a map of that environment using single or multiple sensors. The primary distinction between VO and SLAM is that VO focuses

on local consistency and aims to incrementally estimate the agent's path, pose by pose. SLAM seeks a globally consistent estimate of the agent's trajectory [25]. This section covers a review of the current state-of-the-art VO and SLAM methods.

2.2.1 Visual Odometry (VO)

VO methods can be classified into two types - (1) Knowledge-based, and (2) Learning based [26]. Knowledge-based methods exploit geometrical relations between frames to assess motion, and Learning based methods rely on Machine Learning (ML) techniques.

Knowledge-Based Techniques

Knowledge-based techniques can be further classified into three categories - (1) Appearance-based, (2) Feature-based, and (3) Hybrid. These classifications are created based on how visual components are used to generate odometry estimates [27]. Appearance-based techniques operate on the intensity values of the pixels directly and match the template of sub-images over the optical flow values to estimate motion. Feature-based techniques extract interest points that can be tracked with the help of vectors that describe the local region around the key points. Hybrid methods make use of techniques used both, in appearance-based and feature-based methods.

Feature-based methods work under the assumption that prominent points or regions in each frame can be used to determine camera movement. These key points include lines, edges, corners and other image patterns that can be distinguished from their surroundings in terms of intensity, color, or texture and are thus likely to match well across multiple images [28, 29, 30]. One might argue that this is a two-headed arrow – while this approach works well in cases where image features are prominent, the approach fails to provide meaningful results when that is not the case, for example,

asphalt, sandy soil, etc [27]. For feature detection, it is important to determine which feature should be used. The most common feature descriptors are SIFT [31], SURF [32], ORB [33], BRISK [34] and A-KAZE [35]. [36] provides an in-depth comparison between the feature descriptors listed above by evaluating them on the KITTI [37] benchmark. It is found that both geometric distortions and inconsistent lighting are well handled by feature-based VO [38].

There are three methods for estimating motion: (1) 2D-2D, (2) 3D-2D, and (3) 3D-3D. The most common approach is feature-to-feature matching (2D-2D), which takes advantage of the constraints imposed by Epipolar geometry. 3D-2D techniques reduce re-projection errors from 3D-tracked landmarks to the current image frame. Finally, 3D-3D techniques directly compare two sets of 3D points but are generally less accurate than 2D-2D and 3D-2D [24].

Learning-Based Techniques

ML-based VO methods are one of the emerging techniques for motion estimation because they do not require the camera calibration parameters to be explicitly known. Once an input image sequence is provided, a large dataset is used to train a regression or classification model that can estimate ego motion. These methods can estimate translation to the correct scale and are resistant to the noises on which they are trained [27].

Engel, et al. [39] proposes, Direct Sparse Odometry (DSO), a novel visual odometry method based on a highly accurate sparse and direct structure motion formulation, by combining a fully direct probabilistic model with consistent, joint optimization of all model parameters including geometry, represented as inverse depth in a reference frame and camera motion. Yang et al. [40] propose Deep Visual Stereo Odometry (DVSO), a new Deep Learning (DL) based framework that complements DSO, by ex-

tending the capabilities of DSO with a neural network that produces accurate depth estimates. [41] builds on the work in DVSO, by introducing a new CNN, DepthNet, which predicts the uncertainty associated with estimates. Camera poses are then estimated using another CNN, PoseNet. The introduction of DepthNet and PoseNet show an improvement of roughly 10% in select KITTI sequences for trajectory estimation accuracy.

Wang et al. [42] propose an end-to-end supervised method, DeepVO, which focuses on learning feature extraction with proper geometric significance and implicitly modeling motion dynamics over a sequence of frames. However, the results are found to be somewhat unsatisfactory [24]. [43] proposes a DL technique based on optical flow, DeepAVO. The proposed method in [43] relies on a learning-based optical flow extractor, PWCNet. The optical flow is predicted to compute the relative transformation of the camera pose and reconstruct the 3D structures of the scene using triangulation. The optical flow is predicted to compute the relative transformation of the camera pose and reconstruct the 3D structures of the scene using triangulation. These structures are then used to align depth predictions in order to address the issue of scale inconsistency between pose and depth predictions. This, however, impedes the learning process. Huang et al. [44] proposes a new dynamics-aware visual odometry technique, ClusterVO, that can segment dynamic objects while retrieving the trajectory of the camera, and the trajectory of the detected objects. This technique was further extended in [45].

With the adoption of using CNNs for depth estimation in [40], there has been immense progress in the domain of supervised monocular depth estimation [46, 47, 48, 49, 50, 51]. [51] inherits all loss functions in the self-supervised monocular depth method proposed in [52, 50] and leverages velocity whenever possible to achieve scale awareness. The proposed self-supervised depth estimation network outperforms all

existing depth prediction networks to date.

2.2.2 Simultaneous Localization and Mapping (SLAM)

Lu et al. [53] propose a basic graph-structured model for SLAM, Graph-SLAM which finds the agent pose in an area based on agent motion and observation data. Graph-SLAM also proposes a loop closure system for further optimization and recognizing previously visited locations. Another SLAM technique that uses Extended Kalman Filter (EKF) for nonlinear filtering is introduced in [54]. Montemerlo et al. [55] proposes another approach for localization and uses the Monte Carlo algorithm, however, the algorithm does not work well in modern environments with a higher number of objects [56].

ORB-SLAM2 [57] is a well-known algorithm in the VO and Simultaneous Localization And Mapping (SLAM) communities. It is an extension of ORB-SLAM [58] for monocular, stereo, and RGB-D cameras. This open-source method is widely regarded as a gold standard in visual odometry. As a SLAM technique, this algorithm is made up of three threads: tracking, local mapping, and loop-closing. Motion estimation is computed using ORB features tracked over keyframes and a local map, with a focus on multi-step Bundle Adjustment. Other popular methods for SLAM include VISO2 [37], LSD-SLAM [59], CubeSLAM [60], and SLAM++ [61]. [37] introduces the construction of 3D maps using stereo cameras by applying stereo-matching to a sparse set of features in conjunction with an odometry method that uses Kalman Filter. Engel et al. [59] propose a direct monocular SLAM called Large-Scale Direct (LSD) monocular SLAM. The model takes into consideration the depth estimate for each pixel and then uses the image pixels in the dense map instead of extracted features. CubeSLAM builds on top of the work in [44] and [58] by tracking higher-level objects rather than key points alone. This is achieved by integrating the size and location of

a detected cuboid in graph optimization constrained by a motion model.

2.3 Vehicle Detection & Tracking

Object pose detection and tracking have recently gained popularity due to their widespread application in a variety of fields, most notably autonomous driving and robotics. DL is the most promising technique for object pose detection and tracking, with promising results. The majority of early efforts in this domain relied on expensive LIDAR systems like Velodyne. Because of the input constraints imposed in this manuscript, it is not possible to retrieve high-quality LIDAR data, so the proposed solution must rely on single-camera techniques for vehicle detection and tracking. This section covers techniques used for monocular vehicle detection and tracking.

For vehicle detection and tracking, there are mainly two approaches, the first approach relies on generating 3D bounding boxes directly (using DL), using known geometry of shapes, or computing distances via 2D/3D constraints. The second approach applies a representation transform such that we obtain a Birds-Eye-View (BEV) of the scene and DL techniques can be applied for object detection or segmentation.

First Approach: Estimating 3D Bounding Boxes

Detecting 3D objects directly from 2D images is a difficult task, as depth information is suppressed during the creation of the 2D image. Mousavian et al. [62] proposed a new method for 3D bounding box estimation using DL and geometry, Deep3DBox, that uses discrete continuous loss, which has become a standard for regression of a large range of multi-modal regression problems. Following this approach, there have been various works that utilize the 2D/3D geometry constraints for 3D bounding box estimation [63, 64, 65, 66, 67, 68].

Li et al. [63] proposes an approach that regresses the 2D bounding box and orientation using conventional 2D object detection frameworks and then refines it to get a 3D position. Lu et al. [69] extends this approach and uses the height of the car for depth estimates. Liu et al. [64] build on the approach in [62] by adding a refinement stage that densely samples around the 3D seed location, and then scores the 2D patches with 3D wire-frames. The 3D proposal stage has a refinement model where-in the depth is calculated using intrinsics and the estimated bounding box. The location is then computed by re-projecting the center of the 2D bounding box to the computed depth. Naiden et al. [66] further extend the work in [62] by proposing a method to regress residual center positions. Ku et al. [65] propose another approach, MonoPSR, which generates the 3D proposal first and then reconstructs the local point cloud of the object. Choi et al. [68] extend [62] by building 2D-3D constraint optimization into the neural network and using an iterative method to refine the corner cases.

Chen et al. [70] proposed the first framework, Mono3D, that directly estimated bounding boxes from a single monocular image. Mono3D made use of Faster RCNN [71] backbone for detection and required a 2D segmentation mask generated by SegNet [72] to generate 3D proposals. The 3D proposals are generated based on the premise that cars are on the ground plane. Following this approach, there have been a lot of improvements in other works [73, 74, 75, 76, 77, 78, 79, 80].

Xu et al. [73] proposed the idea of multi-level fusion, the approach concatenates the depth estimation from RGB to RGBD and then performs object detection on the generated image. [73] also investigates using stereo camera images as input and shows that the precision, in that case, is better than mono, showing that the limitation still stands at depth estimation, as shown in [81].

Hou et al. [74] build on previous work done in Deep3DBox [62] by regressing

inverse distance and training the model with RoIAligned [82] features instead of image patches as in [62]. This approach first performs monocular 3D object detection and then applies tracking on it, for an end-to-end monocular 3D object detection and tracking framework. However, it requires the availability of an Inertial Measurement Unit (IMU) or GPS for extrinsic parameter calculation.

Zhou et al. [75] propose a new object detection framework, CenterNet, that builds on the premise of object detection of the center point of an object and regression of the associated properties. It is the first real-time anchor-free object detection framework. Jorgensen et al. [77] builds on CenterNet and parameterizes 2D and 3D bounding boxes in 26 attributes per object. These attributes are then regressed for 2D and 3D bounding boxes. However, this approach requires really accurate intrinsics for meaningful results. The same authors propose a new framework, QD-3DT [80], that tracks moving objects over time and estimates their full 3D bounding box information from a sequence of 3D images from a moving agent. The proposed framework utilizes quasi-dense similarity learning to identify objects from visual cues and associates them in 2D. Furthermore, an LSTM-based object velocity learning module aggregates the long-term trajectory for more accurate motion exploration. The 3D tracking pipeline achieves an improvement of nearly 500%. The method proposed in [80] stands as the current State of the Art (SOTA) for 3D object detection and tracking for vision-only methods.

Qin et al. [78] apply a different approach for monocular 3D object detection. The approach proposes using 3D anchors to explicitly construct object-level correspondences between regions of interest in stereo images and then training a Deep Neural Network (DNN) that learns to detect and triangulate the object in 3D space. Brazil et al. [79] uses yet another approach, where 3D and 3D bounding box parameters are regressed simultaneously by pre-computing 3D mean stats for each 2D anchor.

The proposed method achieves a 5% increase in accuracy among methods until 2019, however, takes a longer time to process rendering it unsuitable for real-time application.

Second Approach: Estimating Birds-Eye-View (BEV) of the Scene

The main challenge of estimating the BEV of the scene is view transformation. The most popular approach for this task is Inverse Perspective Mapping (IPM) [83]. However, IPM assumes that the ground is flat, and as a result, it does not work well in the case of surfaces that are not flat. DL has proved to be a go-to solution for the task of BEV scene understanding, and there has been a lot of work in this area [84, 85, 86, 87, 88, 89, 90, 91, 92].

Pan et al. [84] propose the first network for BEV semantic segmentation. The approach uses a transformer module to model the view transformation and is implemented as a multi-layer perceptron. Hendy et al. [85] builds on the work in [84] and proposes another approach that converts sensor information from LIDAR, RADAR, and camera to a unified representation in BEV. The approach uses a similar view transformation network as in [84] and considers the previous 5 frames from all the sensors to predict 5 frames into the future.

Lu et al. [86] propose a new approach for view transformation, where-in a Variational Auto Encoder (VAE) is used with sampling for view transform. However, the approach is unable to generate sharp edges in the segmentation result.

Some works also tackle the problem of occlusion - Schulter et al. [87] use a pixel-wise depth prediction network for view transformation. The approach additionally learns to predict occluded portions in the image, and it learns general road structure by aligning geo-tagged images to data from OpenStreetMaps while training. Reiher et al. [88] uses IPM for view transformation and relies on a surround camera system

for generating a 360-degree BEV. The approach described in [88] uses synthetic data generated in CARLA for training, pre-processes the training data to introduce the effect of occlusion, and then is trained on the augmented data.

There has been a lot of work that utilizes a surround camera system for BEV. Roddick et al. [89] uses a dense transformation layer for view transformation, and proposes an end-to-end framework that estimates ground plane, and performs road segmentation, lane detection, and 3D object detection. Philion et al. [90] builds on work from [89, 81] and proposes a probabilistic pixel-wise depth prediction network. The proposed method works in three stages, Lift, Splat and Shoot.

2.4 Map Generation

Automatic map generation in the context of this manuscript refers to the process of creating a 3D map of the area in which the scenario takes place. This includes the corresponding road network, road objects, and buildings. This is a crucial task in the domain of automated driving. There are mainly two methods of automatically generating maps.

The first method relies on the onboard vision sensors of the agent for semantic scene understanding. The result can be then stitched together for a complete map [87, 89, 93, 94, 95, 96]. Mani et al. [93] propose an approach, MonoLayout, that uses VAE view transformation, and uses one encoder, and two decoder network architecture. The network also learns road geometries extracted from the OSM database. Li et al. [96] focus on the prediction of vectorized map elements in BEV for scene understanding. Li et al. [96] make use of neural feature transformation and geometric projection for view transformation. Most approaches for scene understanding rely on well-calibrated surround camera systems for meaningful results, and thus cannot be used under the constraints imposed by this manuscript.

The second method relies on the availability of open-source map databases such as OpenStreetMaps (OSM). Mondal et al. [97] propose some ideas for constructing virtual worlds in CARLA using the OSM database. The proposed approach uses Blender to generate world mesh, however, details of implementation are not provided. Maierhofer et al. [98] introduce a toolbox, CommonRoad, that provides converters for different formats of road networks, including OSM, Lanelet, OpenDRIVE, and SUMO. Cai et al. [99] propose extensions to [15] to utilize OSM data to develop and test crowd-driving algorithms. Additionally, CARLA [15] provides a method of generating road networks defined in the OpenDRIVE standard.

Chapter 3

PROBLEM DESCRIPTION

The primary focus of this thesis is to create an end-to-end video processing pipeline that can generate high-fidelity simulation scenarios using monocular dashcam videos with known or unknown camera parameters and trajectories. The scenarios can then be used to test Automated Driving Systems and to collect valuable collision data that can be used to train the ADS. Figure 3.1 shows a single frame from a randomly chosen YouTube video and the resulting simulation in CARLA. The proposed solution works under a set of constraints and assumptions as listed in the following sections.

Figure 3.1: Side-by-Side Comparison of a Frame From Video and Its Corresponding Result in the Simulator

3.1 Constraints

- **Constraint 1**: If the camera parameters are not specified, the video should consist of lane markings, as they are used to estimate the necessary parameters.
- **Constraint 2**: If no reference GPS trajectory is provided, the user must specify a starting GPS coordinate (latitude, longitude, and bearing) in order to create meaningful replays.

- **Constraint 3**: If the user chooses to extract GPS trajectory using the video stamp, the coordinates must either be in float point values or in NMEA format.
- **Constraint 4**: The video should have good visibility and the camera placement should capture the road adequately.
- **Constraint 5**: Videos captured at night time should not have any screen reflections.

3.2 Assumptions

- **Assumption 1**: The camera is approximately at the center of the dashboard.
- **Assumption 2**: OpenStreetMaps data is accurate and reflects the latest road conditions.
- **Assumption 3**: Make and model of other cars are ignored, and cars with similar shapes and sizes are spawned.
- **Assumption 4**: In the generated map, the lane width is 4 meters, and the sidewalk is present along the entire route.
- **Assumption 5**: The ego vehicle is a standard-size sedan (similar to Tesla Model 3).
- **Assumption 6**: The ego vehicle stays on the road throughout the duration of the scenario.

3.3 Line of Approach

To generate high-fidelity simulations, the proposed pipeline follows a modular architecture, where-in, each module is responsible for an independent task such as

ego-motion estimation, trajectory extraction, road user detection, road user tracking, and replay. The estimated trajectories are then simulated in CARLA within a high-definition map generated using data from OSM. The following chapters discuss the internal methodology for each of these tasks.

Chapter 4

TRAJECTORY EXTRACTION

The main function of the trajectory extraction pipeline is to be able to capture the trajectories of the ego vehicle and the surrounding road users. Surrounding road users consist of other vehicles, pedestrians, and cyclists. This functionality is implemented using a combination of current State-of-the-Art deep CNNs and post-processing techniques.

For clean implementation and ease of modification, the pipeline is broken down into 5 stages, wherein the first stage is setting up the system using required inputs, and the following 4 stages are processing stages - (1) Camera Calibration, (2) Ego motion extraction, (3) Vehicle detection & tracking (4) Asset selection & scene setup. Each processing stage relies on the result from the previous processing stages. In the end, the pipeline produces (1) trajectories for ego vehicle and road users in global frame coordinates, (2) an asset map that determines which CARLA asset should be utilized for a road user and the corresponding color, and (3) a scene setup, which contains information regarding the time of day, weather, and more.

All processing stages can be independently thought of as a separate sub-problem. The next sections describe the problem statement, functionalities, requirements, and techniques used to solve the sub-problem. Figure 4.1 represents the high-level processing architecture for the trajectory extraction pipeline. The system accepts video or image frames along with an options structure as input. The setup stage is responsible for reading and parsing GPS trajectories from different formats. The first processing stage computes the camera calibration parameters that are used in the later stages. The second processing stage takes the image frames and camera calibration parame-

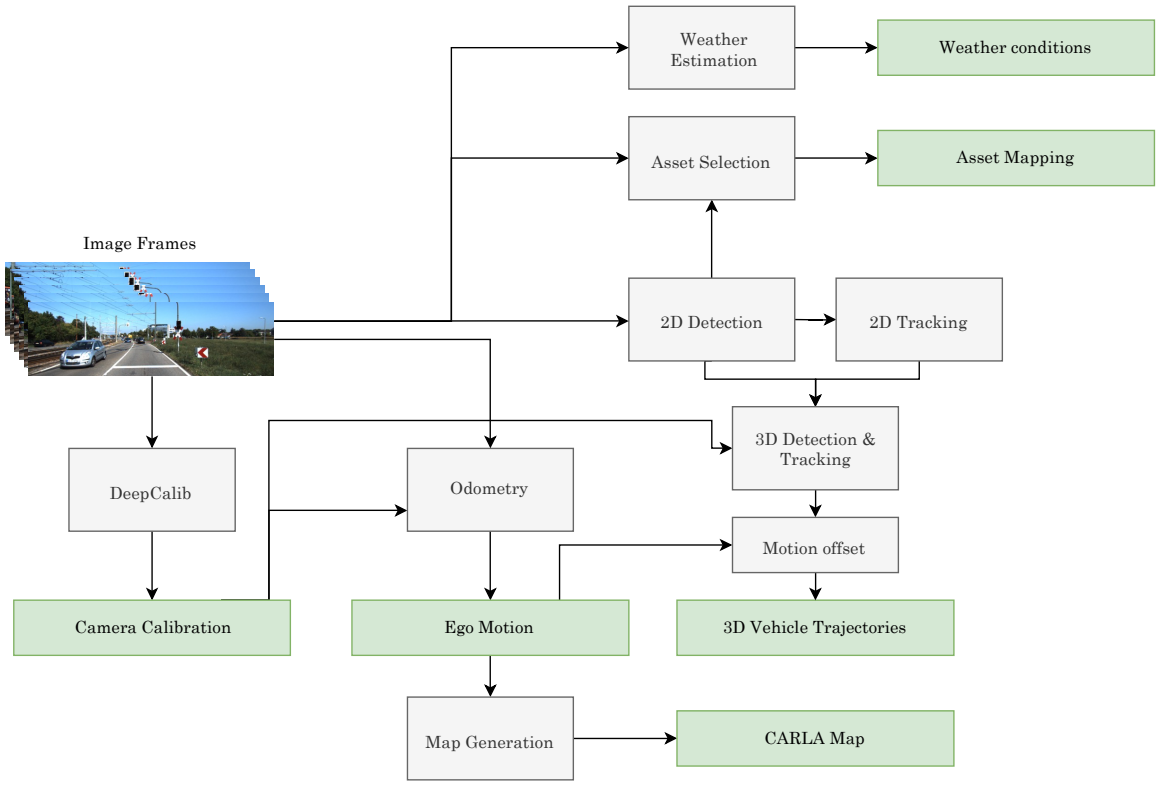


Figure 4.1: Trajectory Extraction: High-Level Architecture

ters to estimate the ego-motion trajectory using VO or SLAM. The third processing stage uses the estimated ego-motion trajectories and computes surrounding road user trajectories in global frame coordinates. The final processing stage creates an asset map that is used by the trajectory simulation stage to determine which CARLA asset should be used for a particular road user. Additionally, the final processing stage also estimates the weather conditions for a more accurate scenario replay.

4.1 Setup

The toolbox is designed in a manner that allows users to choose from different techniques to solve a certain sub-problem. This functionality is introduced for multiple reasons. First, it enables the user to choose techniques that work best with data they are working on. Second, it allows for further extension of the toolbox where-in

a 3^{rd} party developer can easily add techniques for a certain stage in the process.

The toolbox accepts video files or a sequence of images, sorted in alphabetical order as the primary input for trajectory extraction. In the case that the user opts for a video to be the input to the system, the setup stage reads the EXIF information to extract metadata such as total length, frame rate, start time, etc. Additionally, the first step in the pipeline is to extract all the frames from the video and assign a timestamp to all the images. If the timestamp for the video is not present in the EXIF data, then the setup stages use system time to assign a timestamp to the first frame, and then the time difference is interpolated based on frame rate. In the case that the user opts for a sequence of images to be the input to the system, the user also has to specify the frame rate at which they were captured. This is later on used by the ego-motion extraction stage for the computing time difference and speed profiles for each vehicle.

Additionally, the system accepts an `Options` object that allows users to specify the GPS trajectory source, camera calibration parameters (if available), and choice of technique for ego-motion estimation if the GPS trajectory should not be used as a reference for ego-motion.

The toolbox allows for ego-motion trajectory to be provided during setup or to be computed later during the processing stages. Figure **??** represents the internal architecture of the setup stage. In essence, the user has four options to choose from when providing the relevant GPS trajectory. The provided GPS trajectory can be of two types - (1) Full (frame-by-frame), (2) Sparse. If a frame-by-frame GPS trajectory is provided by the user, then that is used as a complete reference for ego-motion. If the user provides a sparse GPS trajectory, then points in between are interpolated with the road geometry in consideration. This brings a limitation that the ego vehicle has to be on the road throughout the scenario as described in Assumption 6 of section

3.2. The GPS trajectory plays a major role in the map generation pipeline as the real-world coordinates are used to fetch information such as buildings, road geometry, road type, satellite imagery, and more.

As explained above, the user has four options to choose from for providing the GPS trajectory.

1. **Embedded**: Most dashcams with GPS capabilities store the GPS information inside the video file. In traditional methods, this information is read by their proprietary tools for video display with the real-time location. A few manufacturers offer the option to export this information to a different format such as CSV or JSON. This toolbox has the capability of reading embedded GPS information right from the video file. The file is read as a byte buffer. Each byte array is then matched with a pre-defined signature for GPS information. Once a byte buffer matches the signature, it can be parsed using a pre-defined format for reading relevant information. Most dashcams store GPS coordinates, speed, bearing, and accelerometer values at the rate of 1 point per second.

2. **Stamped**: In a few cases, it may not be feasible to have raw videos from dashcams such that embedded information can be accessed. However, a lot of videos have the coordinates stamped on the video in a particular format (NMEA / Float). The stamp is generally found to be in the same position throughout the video. If the user opts to use the stamped formation, the user has to select a region of interest for latitude and longitude respectively. These regions are then read using Optical Character Recognition (OCR) techniques to extract the required coordinates. There are many techniques and state-of-the-art tools available for the task of OCR. However, in this context, the stamps are generally in a cleanly typed font and thus tesseract provides sufficiently good results. If

required, tesseract can be swapped with another model in the future.

3. **External Files**: The system supports reading GPS trajectories from external files, such as GPX trace files, CSV files, or JSON files. The input CSV or JSON has to follow a certain structure, and it can be found in the documentation of the tool. The setup stage automatically determines if the provided file contains full or sparse trajectories, and proceeds further.

4. **None**: While the overall performance can be affected, the toolbox does offer a method to run without providing corresponding GPS trajectories. In the case that no GPS trace is provided, then the user has to input a starting latitude, longitude, and bearing for the simulation to proceed. The GPS trajectory computed by SLAM or odometry methods in the next processing stage is then fit to the provided starting location for replay. If the user does not know where exactly the simulation is taking place, they can feed the input for a location with similar road structures, however, the mesh, and imagery around the place will be significantly different while trajectory simulation.

4.2 Stage 1: Camera Calibration

Camera calibration is a necessary step in 3D computer vision in order to extract metric information from 2D images. 3D vehicle detection and tracking require proper camera calibration and fine-tuning for the best results. In situations when the camera calibration parameters are available, the toolbox offers support to use the pre-calibrated parameters instead of computing the parameters during processing. Figure 4.2 represents the overall architecture of the camera calibration stage. If the user does not have access to the camera calibration parameters, they can be estimated using one of the two supported techniques.

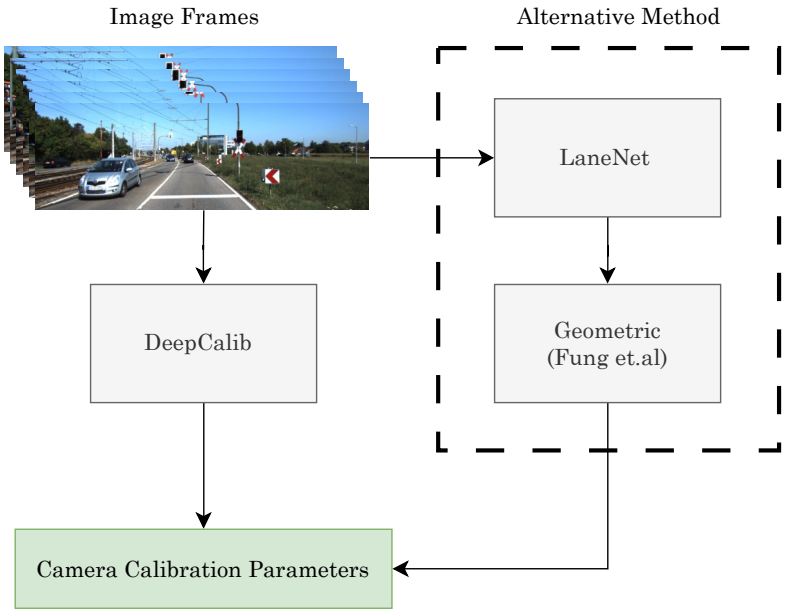


Figure 4.2: Trajectory Extraction: Camera Calibration Stage

1. **Geometric**: A simple and light-weight approach as shown in [22] is used to reformulate the perspective camera equations. Lane line annotations are then used to estimate the camera parameters. For this method to provide sufficiently good results, the lane lines should be predicted very accurately. For ensuring that the geometric camera calibration system is using accurately predicted lane lines, the toolbox launches a visual window during the processing stage giving the user a choice in terms of which frame should be used for focal length estimation. Additionally, the toolbox also provides the user to manually select four points on the image such that a quadrilateral formed with those four points is a perfect rectangle when seen from a top-down perspective.

2. **DeepCalib** [100]: A deep CNN trained on the SUN360 [101] dataset using the SingleNet classification architecture. This approach computes the focal length on 30 random frames from the video and takes the average focal length. This method also assumes that the principal point is at the center of the provided

image and then constructs an intrinsic camera matrix from the estimated focal length.

As explained above, for the geometric technique of estimating the camera parameters, it is essential that the lane lines are predicted accurately. The camera calibration stage uses LaneNet [102] due to its simplistic architecture and high generalization capabilities. Furthermore, if the user does not make a selection for the preferred method while setting up the toolbox, DeepCalib [100] is used for camera calibration.

4.3 Stage 2: Ego Motion Estimation

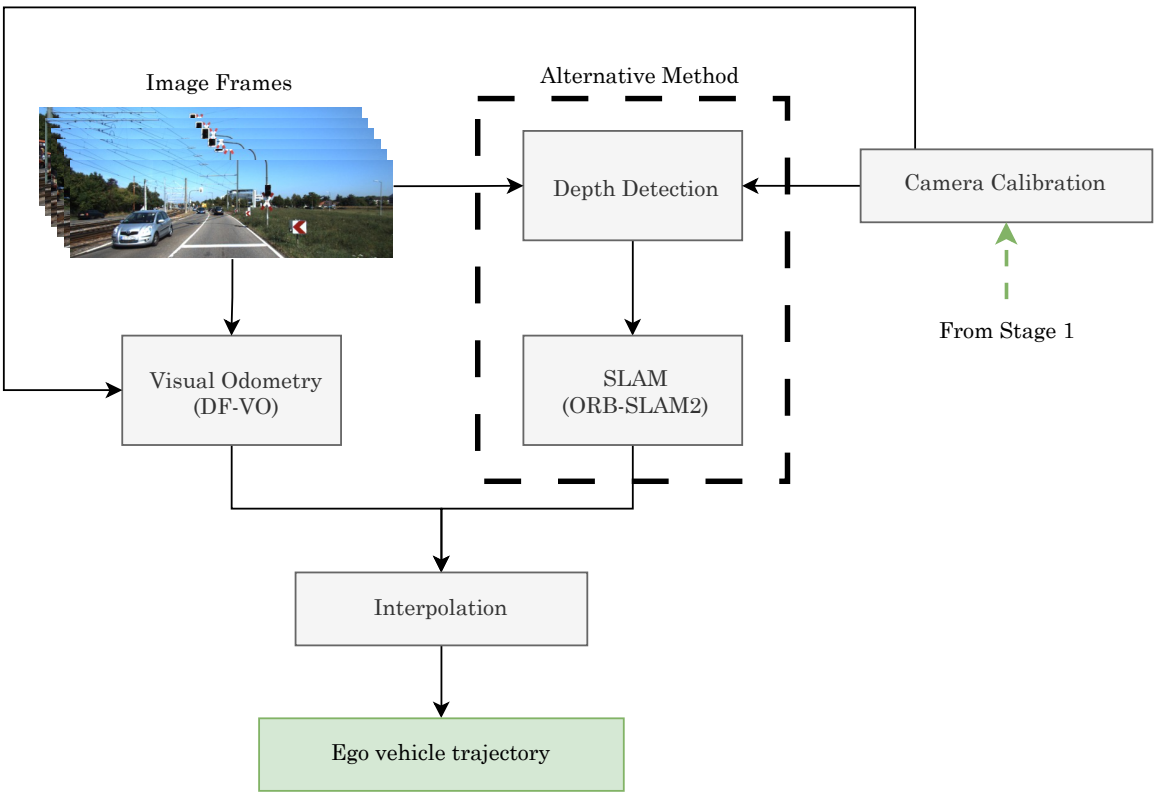


Figure 4.3: Trajectory Extraction: Ego Motion Estimation Stage

Accurate ego-motion estimation is one of the most important aspects of the toolbox, as it is extensively used by the further processing stages to align road user

trajectories with respect to the ego frame and then with the global frame. If the user provides the ego-motion trajectory during the setup stage of the pipeline, then those trajectories are interpolated and used by the system as ego-motion. However, if the user does not provide ego-motion trajectories, then they can be computed using one of the two techniques supported by the toolbox. Figure 4.3 represents a high-level overview of the ego-motion stage.

1. **SLAM**: If the user chooses to use the SLAM technique for ego-motion estimation, then the toolbox uses ORB-SLAM 2 [57] internally in the RGBD configuration. ORB-SLAM 2 is chosen as the method performs frame-by-frame feature matching and supports matching to a local map to identify the real-time location of the agent. It also supports bundle adjustment for minimizing error when estimating motion trajectories. Additionally, ORB-SLAM2 uses pose-graph optimization to minimize drift and supports loop closure, where-in the algorithm can detect when the agent returns to a previous location and uses this information to reduce uncertainty in map estimates.

   Figure 4.4 represents the overall processing pipeline for ORB-SLAM2. The pipeline consists of three parallel threads for tracking, local mapping, and loop closure. The tracking thread pre-processes the image input so that the rest of the system can operate independently of the input sensor.

2. **Visual Odometry**: If the user opts for the VO backbone for ego-motion estimation, then the toolbox uses DF-VO [103] internally to estimate ego-motion. DF-VO is chosen as the method samples high-quality correspondences from deep optical flows and recovers accurate camera poses with geometric constraints. DF-VO also addresses the scale drift issue by geometrically triangulated depths in dynamic scenes.

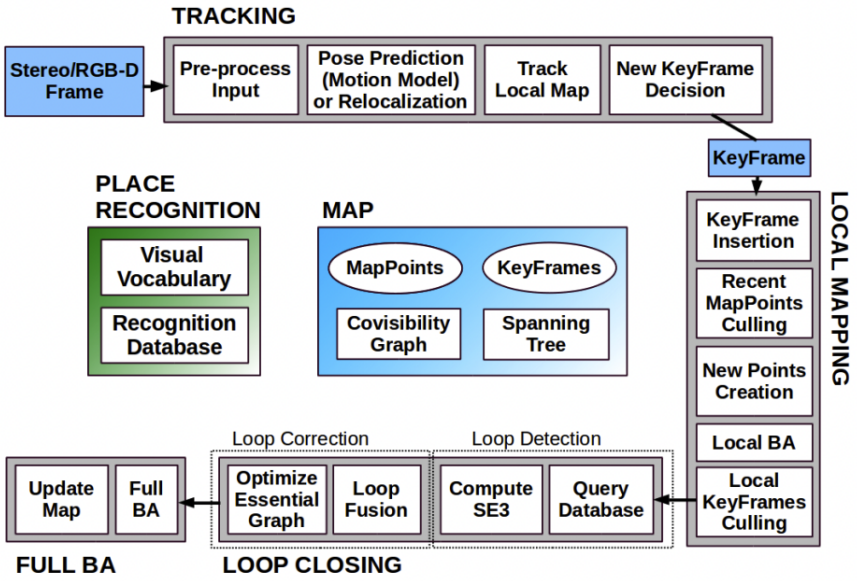


Figure 4.4: ORB-SLAM2: Processing Pipeline [57]

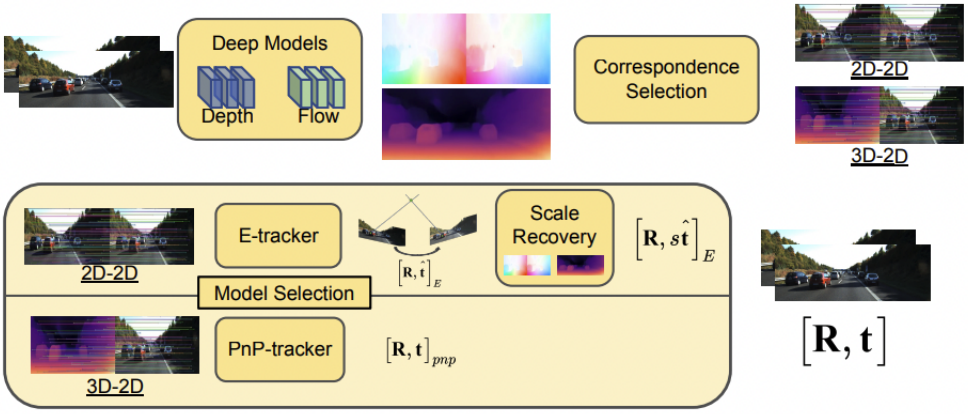


Figure 4.5: DF-VO: Processing Pipeline [103]

Figure 4.5 represents the overall processing pipeline for DF-VO [103]. The pipeline considers an image pair that consists of the frame at time t and a frame at time $t-1$ and then predicts the optical flow and single view depths for each image. It then computes a forward-backward flow consistency to identify and filter good 2D-2D and 3D-2D correspondences. Generally, the top-N flows with the least inconsistency are used for estimation. In the further processing stages,

there are two trackers (1) E-tracker and (2) PnP-tracker. E tracker is used as the primary tracker to track general motion using 2D-2D correspondences. The second tracker uses single-view depth estimates to estimate motion when the E-tracker fails.

If the user does not specify a certain backbone, then the Visual Odometry pipeline is used as it performs on par with ORB-SLAM2 and works in real-time.

4.4 Stage 3: Vehicle Detection & Tracking

The problem statement of extracting trajectories for surrounding road users can be reformulated as 3D object detection and tracking. The end goal is to be able to generate 3D trajectories of each road user such that they can be replayed in CARLA.

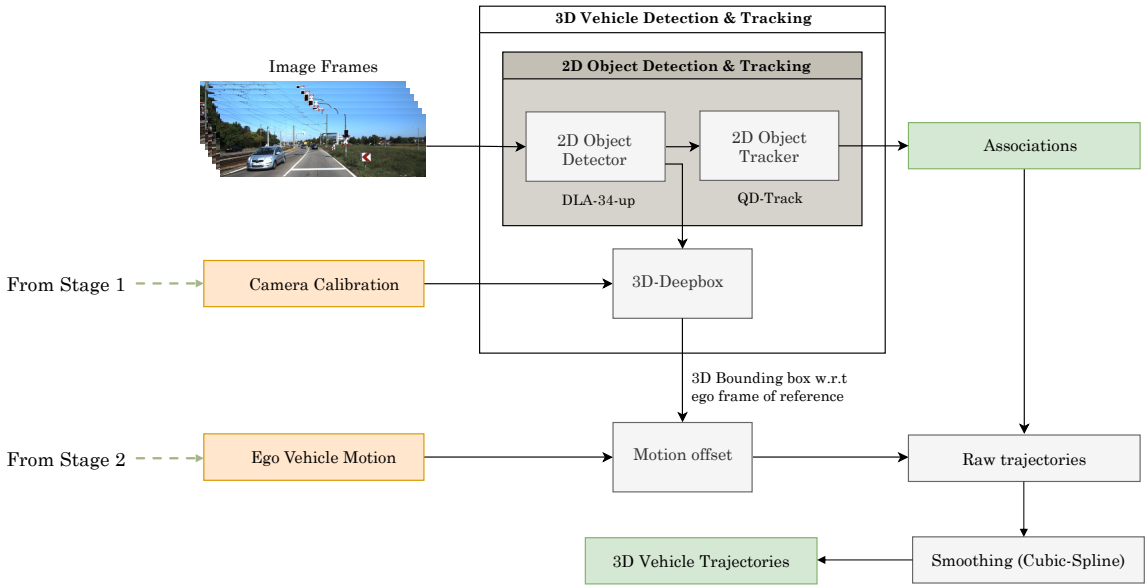


Figure 4.6: Trajectory Extraction: Vehicle Detection & Tracking Stage

As represented in figure 4.6, the toolbox follows the approach presented by quasi-dense 3D detection and tracking [80] to estimate 3D bounding boxes with respect to the ego frame. This method follows an online approach that proposes regions of interest in 2D, from which a 3D layout is estimated. The 3D layout does not

directly project a 2D center into 3D space and also takes into consideration the depth, dimensions, and orientations of the detected object. This greatly improves performance when road users are truncated. The method also uses an LSTM-based tracker that leverages occlusion-aware association and depth-ordering matching, for the refinement of the 3D bounding box. Additionally, the method scores SOTA results on the Agroverse and KITTI 3D tracking benchmarks.

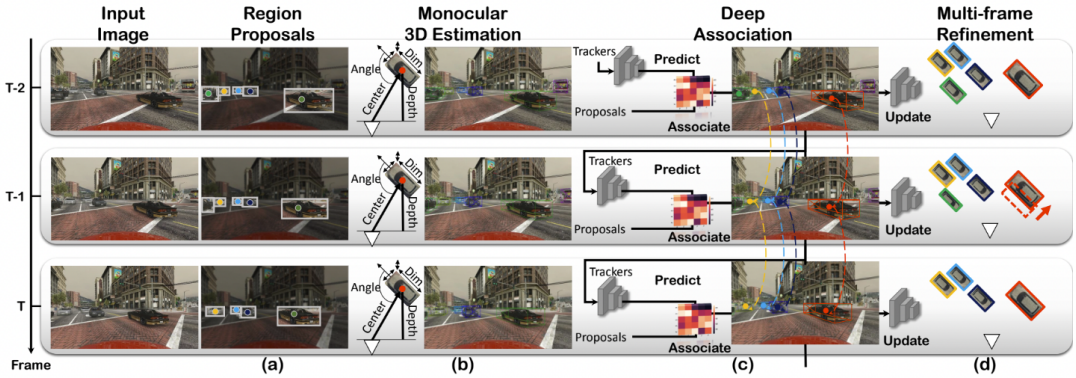


Figure 4.7: Monocular 3D Detection & Tracking: Processing Pipeline [80]

Figure 4.7 represents the internal processing pipeline for quasi-dense detection and tracking. The first stage makes use of a Faster R-CNN [71] to generate 2D object proposals in the form of a bounding box. These bounding boxes are used by the pipeline to extract appearance features but are not used for projecting a 3D center. The 3D center is estimated by using an ROI head with the regression process. The network is able to regress bounding boxes along with the 3D project of the box center. Once the 3D center is available, a 3D box is estimated by extending [62]. The pipeline is capable of regressing inverse depth and is trained with ROI Aligned features instead of image patches as in [62]. Instance associations are generated by depth ordering of trackers before matching using the Hungarian algorithm. Finally, a deep motion model based on two LSTMs is used for predicting vehicle motion and then refining

the 3D bounding box.

After all the 3D bounding boxes are estimated, the next step is to project the location of each bounding box in global frame coordinate using ego vehicle trajectory and apply tracking association such that there is one global trajectory for each object instance. Ideally, the 3D bounding box estimations should work out of the box in terms of creating trajectories, however, due to disparities in depth estimation and imperfections in camera calibration parameters, the resulting trajectories are imperfect and need to be smoothened before simulation. Furthermore, speed profiles need to be computed such that the simulation controller can replay the scenario at the same pace with respect to the input video when using the physics engine.

4.4.1 Global Frame Trajectory

To compute global frame trajectories for all the road users, the pipeline starts with a simple assumption that the ego vehicle trajectory begins at $(0, 0)$. Interpolated ego-motion trajectory is then used as a reference when determining the global trajectory for each road user. The pipeline estimates all the annotations per frame and assigns an instance ID to each unique road user within the video. Additionally, the pipeline has knowledge about where the ego vehicle is in each frame. Now, local coordinates (with respect to the ego frame) can be computed for each 3D bounding box by using the forward and right direction vectors, where-in the forward vector represents how far the road user is from the ego vehicle and the right vector represents the lateral offset from the center of the ego vehicle. After the local coordinates are computed, the ego vehicle location in the global is used to offset the local coordinates for each road user.

4.4.2 Trajectory Smoothing

After extracting global frame coordinates for each road user, the trajectories need to be smoothened such that the generated simulations do not have jerky behavior for road users. The pipeline first filters out trajectories for vehicles that appear and disappear constantly throughout the video, and for vehicles whose, time to live is less than 1-2 seconds (generally around 20-60 frames, depending on the input video). When a vehicle keeps appearing and disappearing, the vehicle cannot be removed from the scene for the frames it is not visible in the front view, and instead, the trajectories for that duration have to be interpolated. This can pose an issue to the system in case the spline function generates trajectories that crash with the ego vehicle if the vehicle is too close. It is important to note that there can be some crash scenarios that will involve the vehicle that keeps appearing and disappearing. For those scenarios, the toolbox offers a method to disable this sort of filtering. After the filtering process, each trajectory is smoothened using a natural cubic spline function.

4.5 Stage 4: Asset Selection & Scene Setup

Asset selection in the context of the work refers to the task of choosing the correct vehicle or model for each object instance. It is an important aspect of trajectory simulation as randomly choosing assets might cause unexpected behaviors. For example, choosing a trailer truck asset instead of a mid-sized vehicle in a congested traffic scenario might cause crashes that do not exist in the original video and thereby affect the overall dynamics of the replay. Along with a module to identify which vehicle should be used for a certain object instance, the toolbox also supports identifying the colors for vehicles. Figure 4.8 represents the overall pipeline for asset selection and scene setup. The next sub-sections cover each aspect briefly.

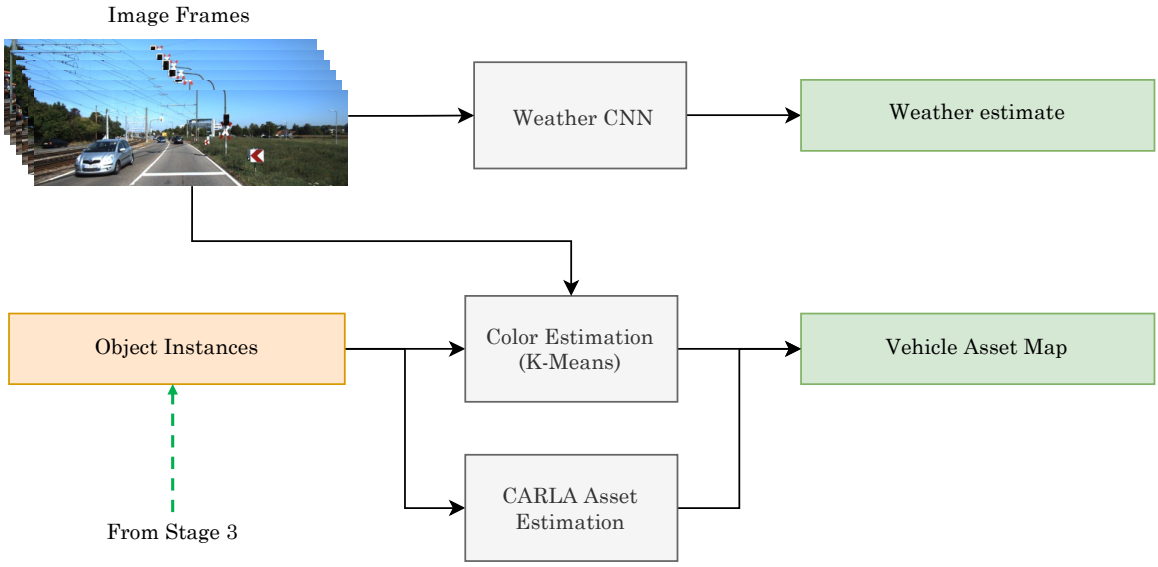


Figure 4.8: Trajectory Extraction: Asset Selection & Scene Setup

4.5.1 Vehicle Color Estimation

The toolbox supports estimating the color of each surrounding vehicle such that it can be used during replay. This is achieved by extracting the 2D bounding box for each vehicle instance and applying k-means clustering to the pixel values within the region of interest to identify the dominant colors. The most dominant color is assumed to be the color of the vehicle in consideration. This approach is not state-of-the-art but is found to perform sufficiently in well-lit scenarios.

4.5.2 CARLA Asset Selection

During the vehicle detection & tracking stage, dimensions for each road user are estimated, and as the car is observed over multiple frames, the estimated dimensions change. The car asset selection stage takes into consideration all the dimension estimates for each vehicle instance and computes the average length and width of the vehicle. To determine which CARLA asset should be used for a particular vehicle instance, an embedding consisting of dimensions for all CARLA assets is pre-computed.

During the selection stage, the nearest (L2 norm) data point in the embedding is queried and the asset corresponding to that point is used during the trajectory simulation stage.

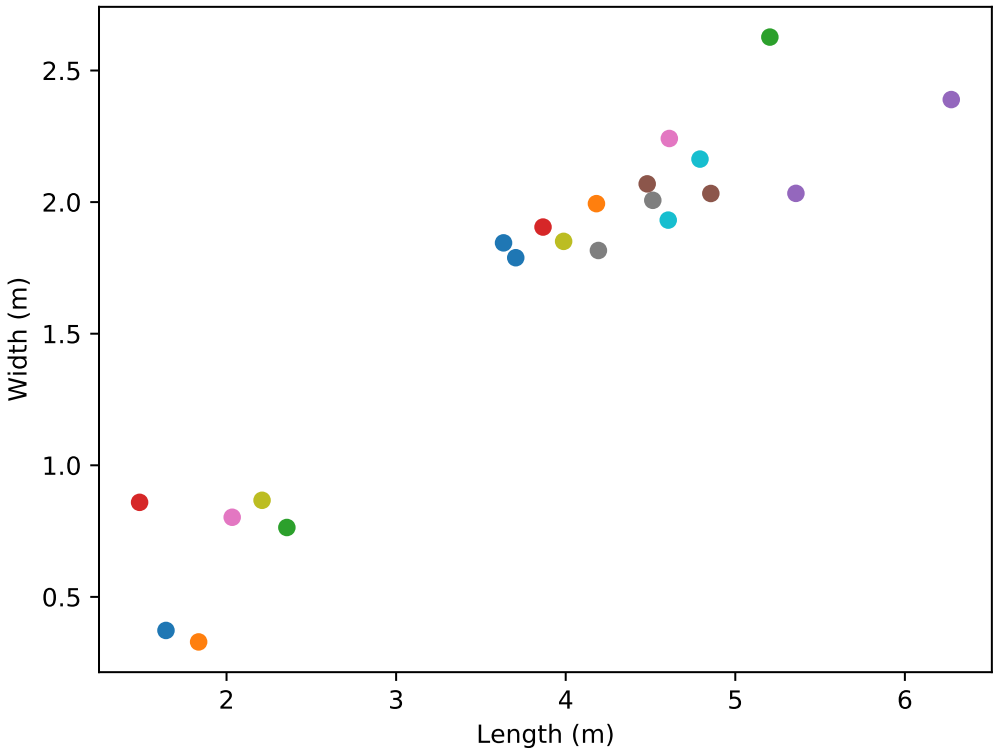


Figure 4.9: Pre-computed Embedding for All CARLA Vehicle Blueprints

Figure 4.9 shows the pre-computed embedding consisting of dimensions for all CARLA vehicle blueprints. There are only 21 vehicle blueprints available, and thus a simple brute-force approach is used for computing the nearest data point. In the situation where the number of assets increases or the number of object instances becomes too large, the nearest neighbor search can be extended to use KD-tree or other fast nearest neighbor algorithm approaches.

4.5.3 Weather Estimation

The toolbox supports estimating the weather conditions using a deep CNN trained on scene tagging dataset from BDD100K [104]. The trained CNN consists of a 34-layer Deep Layer Aggregation (DLA) [105] backbone and can classify as rainy, snowy, clear, overcast, partly cloudy, cloudy, and foggy. To avoid wrongly classifying a scene, the weather estimation stage randomly samples an odd number of frames and classifies the weather in all of them. The most frequent result is then used for scene setup during trajectory replay.

Chapter 5

MAP GENERATION

The primary purpose of the toolbox is to reconstruct scenarios that happen in the real world. To accurately replay the estimated trajectories in a simulation, it is essential that the virtual world within the simulator closely matches the real-world. The map generation stage allows the toolbox to capture any geographical location and automatically generate a corresponding CARLA virtual world by constructing road networks and nearby static objects. Authoring a map manually is a time-taking process and requires a considerable amount of effort. Without an automatic map generation pipeline, the user will have to manually create virtual maps for CARLA to be able to accurately replay trajectories.

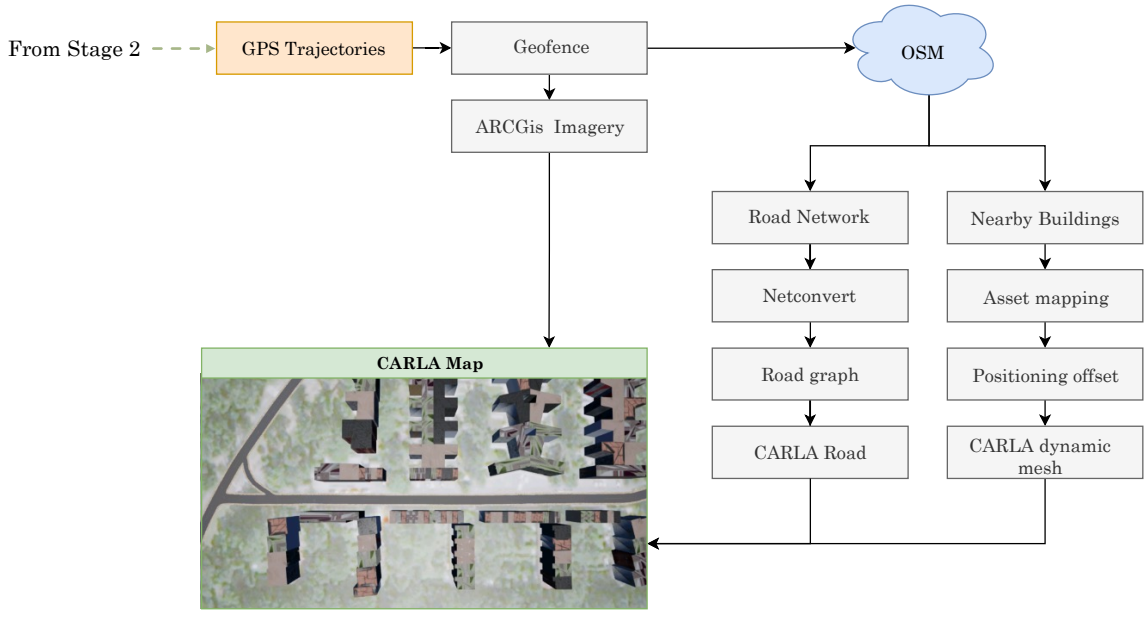


Figure 5.1: Map Generation Pipeline Overview

Figure 5.1 represents the overall pipeline for generating maps that can be used

for trajectory simulation. The pipeline starts off by estimating a geographical region of interest based on a starting GPS coordinate or the entire GPS trajectory. The extracted region of interest is then converted to a CARLA map in three stages - (1) Road reconstruction, (2) Static object reconstruction, (3) Satellite imagery. The sections below describe each of these stages in brief.

5.1 Generating Road Networks

Constructing road networks that accurately follow the geometry of real roads is the most important aspect of the map generation stage. Moreover, it is essential that the generated road network allows the sampling of waypoints such that a controller can be used to follow a specific route.

The toolbox relies on data from the free and open-source map database OpenStreetMaps (OSM) [15] to fetch road and building geometries for a geographical region of interest. The toolbox then internally uses `netconvert` to convert road geometries from OSM to SUMO networks. A SUMO network defines the traffic-related part of a map in the form of a directed graph, wherein each node represents an intersection and each edge represents a lane or a street. The roads are represented as SUMO networks due to the simplicity of representation and availability of an extensive suite of tools that allow editing of the road geometry and structure if required. Additionally, using SUMO network representation allows the toolbox to internally use waypoint sampling algorithms [99]. Using the SUMO network representation, each edge (lane or street) is converted to a polygon in the CARLA world coordinate frame. During simulation, a generic CARLA road mesh texture is applied to each generated polygon. Figure 5.2 shows an example road network generated from KITTI tracking test sequence 4 trajectories.

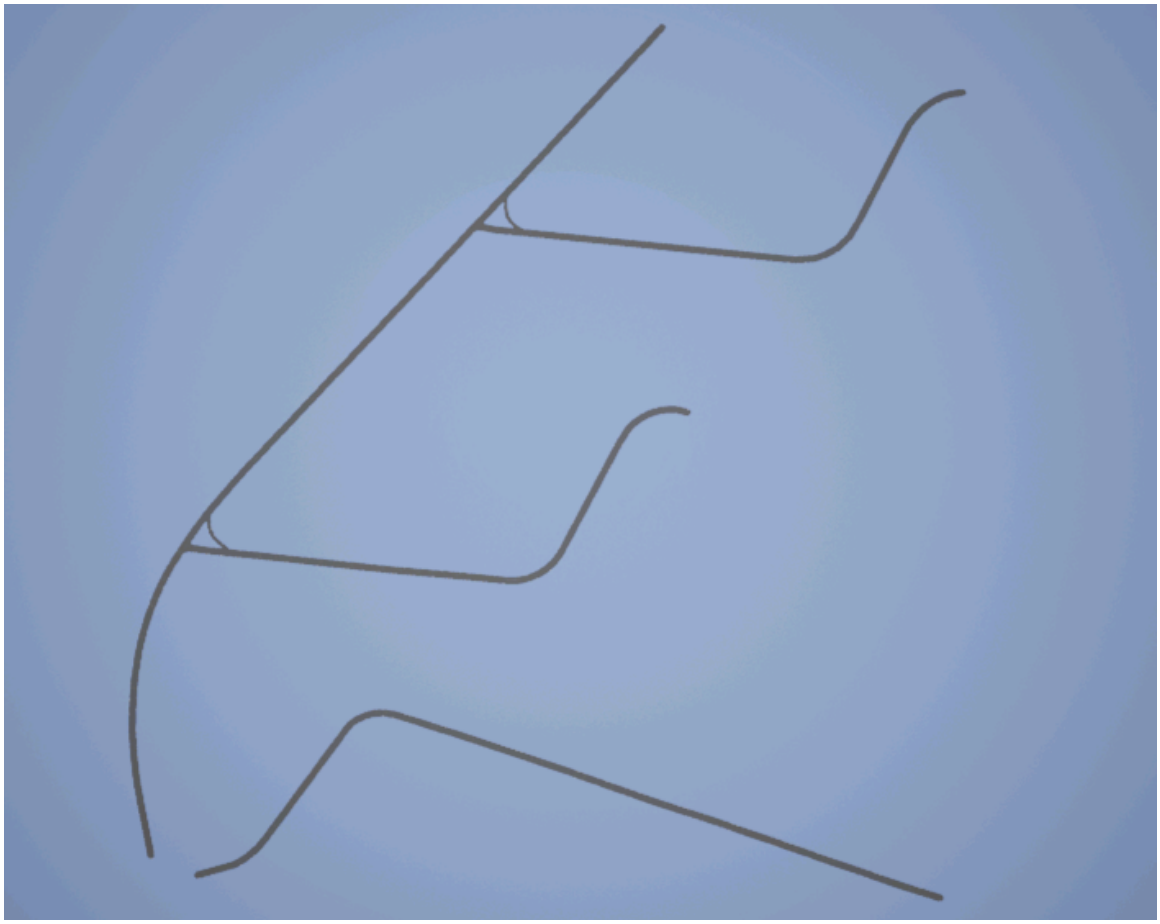


Figure 5.2: Example: Generated Road Network for KITTI Tracking Test Sequence 4

5.2 Generating Buildings

Constructing nearby buildings is not required by the trajectory simulation stage, but adds a finishing touch to the generated maps. Similar to the road generation stage, building geometries are first fetched from OSM, and then converted to polygons in the CARLA world coordinate frames. During simulation, if the building material is available in CARLA, then the corresponding texture is applied to the building. Otherwise, a default black color texture is applied to each building. Additionally, the map generation pipeline also makes use of satellite imagery so that the buildings and the road networks look like a continuous body instead of small assets spread out in

the environment. Figure 5.3 shows an example of a complete CARLA map generated using OpenStreetMaps data and SUMO road networks.

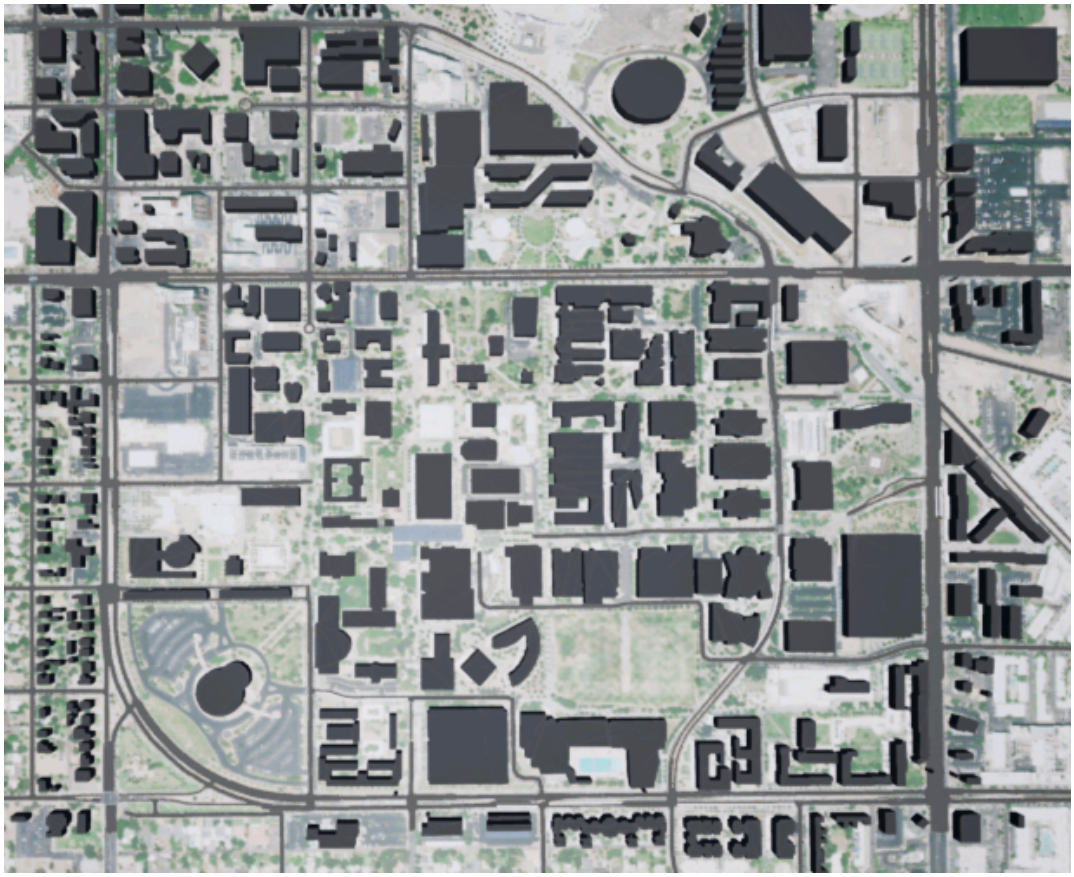


Figure 5.3: Example: Complete Map Consisting of Buildings and Roads for Arizona State University, Tempe, AZ, USA

Chapter 6

TRAJECTORY SIMULATION

The final stage of the toolbox is replaying the estimated trajectories within CARLA to collect crucial traffic analysis information and to create a baseline for further ADS testing. The trajectory extraction stage produces information such as ego vehicle motion and speed profile along with road user trajectories, asset mapping, and weather estimates. Furthermore, the map generation stage uses GPS information to estimate a geographical region of interest, which is then utilized to create a virtual world within CARLA. The trajectory simulation stage utilizes all of the estimated information and creates a scenario that is replayed within CARLA.

The toolbox introduces a Simulation Manager, which is tasked with controlling each and every aspect of the simulation - from scene setup to trajectory replay. The SimulationManager has three main modules - (1) SceneManager, (2) EgoManager, and (3) ActorManager.

6.1 Scene Manager

The toolbox internally uses a scene manager to load and set up the CARLA world for simulation. The scene manager mainly performs three tasks.

1. **Loading the generated map in CARLA**: The first task is to load the generated map into CARLA. This requires the road network geometry, building geometry, and satellite imagery computed in the map generation stage. The scene manager starts off by spawning each section of the road network within CARLA and applying generic road texture. Furthermore, lane divider texture is applied to the road mesh to create lane boundaries. It is assumed that all lanes

are 4 meters in width. After the road network is in place, the scene manager generates each building one by one, by creating a static mesh and applying an appropriate texture (if available) to it. Lastly, to create an effect of continuity between the generated roads and buildings, satellite imagery is placed on ground level within CARLA.

2. **Setup time of simulation**: During the trajectory simulation step, the toolbox allows the user to set a specific time of day for simulation. This is especially useful when recreating scenarios at night-time or during sunsets when the sun is directly facing the onboard cameras.

3. **Setup weather conditions**: The scene manager then utilizes the CARLA weather API to recreate weather conditions as estimated during the asset selection stage in the trajectory extraction pipeline.

6.2 Ego Manager & Actor Manager

The toolbox internally uses ego manager and actor-manager to simulate ego vehicle and surrounding road user trajectories respectively. After the scene manager sets up the CARLA world with required road networks, an ego vehicle (Tesla Model 3) is spawned at a location that is offset slightly corresponding to the interpolated GPS trajectory. The ego manager consists of four main modules that are used to simulate ego vehicle trajectories:

1. **EgoInitializer**: The trajectory simulation stage utilizes the CARLA physics engine for simulating the ego vehicle trajectory. The ego initializer requires the estimated ego-motion trajectory along with the ego speed profile. After the ego vehicle is spawned at the offset location, a PID controller is used to initialize the ego vehicle to its required speed.

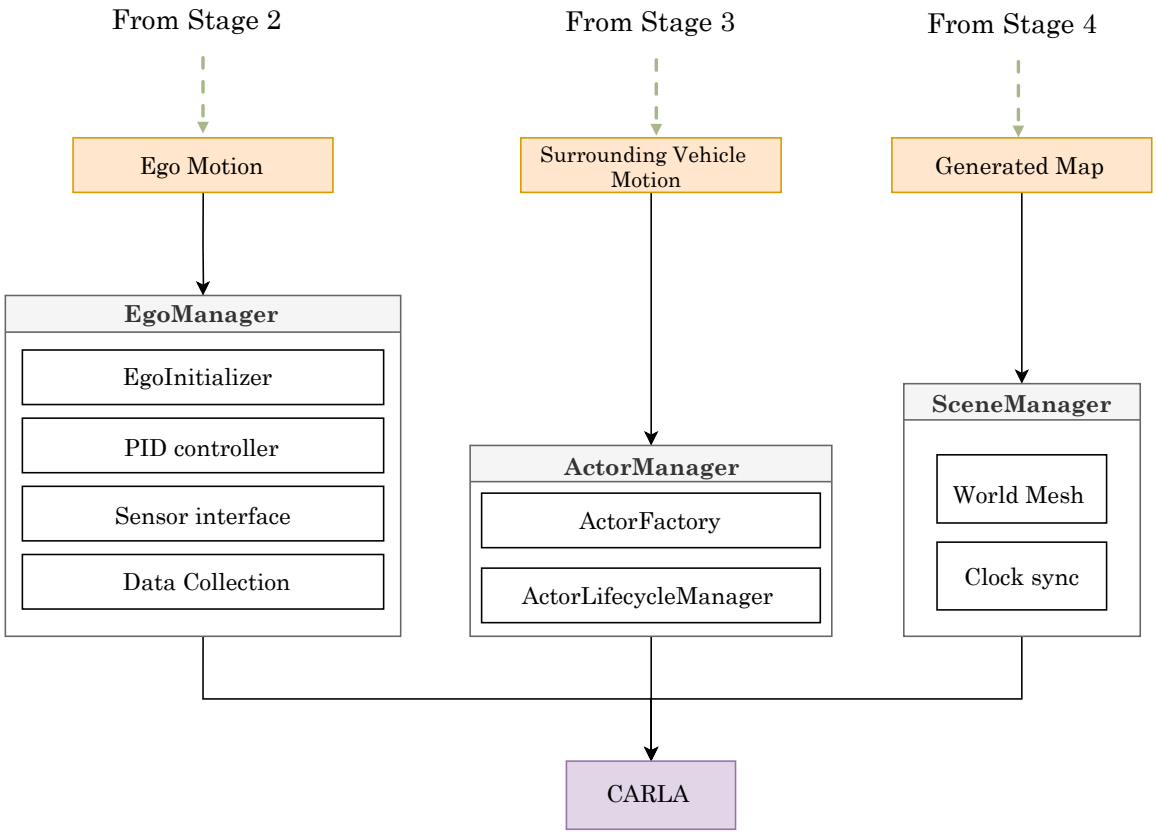


Figure 6.1: Trajectory Simulation: Ego and Actor Manager

2. **PID Controller**: The toolbox makes use of waypoint sampling methods on the SUMO road network to estimate a set of waypoints that the ego vehicle has to follow throughout the simulation. A PID controller is used to estimate vehicle controls such that the ego vehicle follows the pre-determined path.

3. **Sensor Interface**: Multiple sensors such as LIDAR, RADAR, and cameras are attached to the ego vehicle for the collection of meaningful crash and traffic analysis metrics. Additionally, a camera sensor is attached behind the ego vehicle at a certain angle for visualization purposes.

4. **Data collection pipeline**: Data from all the sensors is recorded and exported for further analysis of the scenario.

As the ego manager controls the ego vehicle within the simulation, the actor-manager controls the surrounding road user trajectories. The actor-manager has two main modules:

1. **Actor Factory**: The toolbox uses the actor factory to spawn surrounding road users within the scenario. The module requires asset mapping as estimated during the trajectory extraction stage to determine the correct CARLA asset for each road user. Additionally, the actor factory also paints the vehicles in correspondence with the estimated color.

2. **Actor Lifecycle Manager**: The scenario takes place in steps, and is indexed with respect to each frame from the video. The actor lifecycle manager is responsible for creating, deleting, and updating actors in the simulation with respect to the progress of the simulation.

To summarize, the toolbox internally uses the scene manager to set up the CARLA world using the generated map, estimated weather conditions, and the time of day provided by the user. The toolbox then makes use of ego manager and actor-manager to control the ego vehicle and all surrounding road users respectively.

Chapter 7

RESULTS

The primary purpose of the toolbox is to reconstruct traffic scenarios from videos found on the internet. Generally, these videos do not have ground truth data associated with them, and hence quantitative evaluation is not a straightforward task. Additionally, according to [17], it is important to note that since the primary application of these methods is to create simulations that are being tested in virtual environments that are going to be further modified, there is no need to accurately reproduce quantitatively accurate driving scenarios, and that visual inspection for qualitative correctness is sufficient for the application.

The following sections provide quantitative (wherever possible) and qualitative evaluations for individual processing stages including camera calibration, ego-motion estimation, vehicle detection, vehicle tracking, asset selection, and scene setup.

7.1 Toolbox Performance

This section evaluates the computing performance of the toolbox in terms of processing time based on the number of frames, and the number of vehicles. Table 7.1 summarizes the time taken for the vehicle detection, vehicle tracking and asset selection for 29 test sequences from the KITTI [37] tracking test benchmark. The processing time is computed on a workstation with 20 core Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz, 64 GB RAM, and an NVIDIA Geforce GTX 1080 Ti GPU. From the table, it can be estimated that the processing pipeline requires 1.04 and 3.15 seconds per frame for the vehicle detection & tracking, and asset selection stages respectively. Overall, the entire processing pipeline takes roughly 0.78 seconds per

frame for processing.

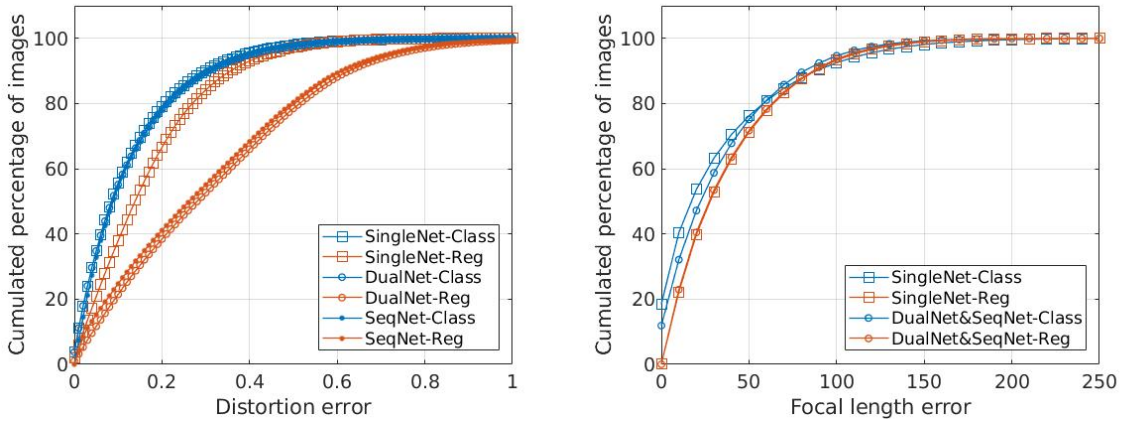
Table 7.1: Toolbox Processing Time on 29 KITTI [37] Sequences

Sequence	**FC** [1]	**VC**[2]	**Processing time (s)**		
			D & T[3]	**AS**[4]	**Total**
0000	465	173	504.4	203.6	710.7
0001	147	9	102.6	10.1	114.8
0002	243	53	200.9	94.1	296.6
0003	257	57	205.2	60.0	266.3
0004	421	25	292.6	28.0	322.2
0005	809	61	527.2	67.9	597.1
0006	114	11	86.6	12.6	100.2
0007	215	41	179.6	44.9	228.0
0008	165	58	154.8	85.8	241.9
0009	349	15	240.6	16.4	259.8
0010	1176	94	874.2	105.6	982.5
0011	774	154	690.9	175.8	869.9
0012	694	185	650.8	216.7	870.8
0013	152	51	152.3	58.9	212.6
0014	850	332	822.6	383.3	1210.4
0015	701	129	639.2	145.8	788.6
0016	510	247	458.7	261.8	723.2
0017	305	52	257.0	59.4	318.0
0018	180	98	199.1	102.3	302.9
0019	404	257	593.2	301.2	896.3
0020	173	75	221.1	76.5	298.8
0021	203	81	263.6	84.5	349.5
0022	436	170	561.7	175.3	739.2
0023	430	150	505.1	151.6	658.7
0024	316	141	466.0	145.8	613.6
0025	176	92	264.0	95.4	360.7
0026	170	96	195.6	96.2	295.6
0027	85	113	106.0	123.0	231.4
0028	175	139	238.8	145.7	385.9

[1]Frame count: Number of frames in each sequence
[2]Vehicle count: Number of vehicles detected in each sequence
[3]Detection and tracking
[4]Asset selection

7.2 Camera Calibration

Camera calibration estimation is one of the most important tasks of the toolbox. Incorrect camera calibration parameters can lead to inaccurate surrounding road user trajectories. The processing pipeline offers the users two backbones for camera calibration estimation - (1) Geometric method using lane line annotations [22] and (2) DeepCalib [100], a deep CNN that is capable of estimating camera calibration parameters accurately using just a single RGB image. The toolbox by default uses a deep CNN trained on the SUN360 [101] dataset using the Single-Net classification architecture. DeepCalib introduces two architectures for estimating the focal length and distortion parameters - (1) Single-Net, and (2) SeqNet. Each of these architectures can be trained for the task of classification or regression.



(a) (b)

Figure 7.1: Camera Calibration: Error Distribution of Estimated Distortion (Left), and Focal Length (Right) Using DeepCalib [100].

Figures 7.1a and 7.1b summarises the cumulative error distribution for distortion and focal length parameter estimations respectively. From the figures, it is clear that the classification architectures for both Single-Net and Seq-Net outperform the regression models, and Single-Net marginally outperforms the Seq-Net architecture.

7.3 Ego Motion

Accurate ego-motion estimation is crucial as it is used to re-project 3D bounding boxes from the ego perspective onto the global frame in order to compute the surrounding road user trajectories. Within ego-motion estimation, the toolbox deals with two main elements - (1) Pose estimation, and (2) Speed profiles. Pose estimation refers to the camera pose trajectory in the global frame, wherein the ego vehicle is assumed to start at (0, 0). The speed profile is then computed using the rate of camera pose change in metric units.

7.3.1 Pose Estimation

Pose estimation in the context of ego-motion estimation refers to the process of computing the camera pose trajectory throughout the input video. When GPS trajectory for the ego vehicle is unavailable, the toolbox offers users to choose from two backbones for ego-motion estimation - (1) ORB-SLAM 2 [57] and (2) DF-VO [103]. This describes the metrics used to evaluate the performance of both the approaches and summarizes their performance on the KITTI [37] Odometry benchmark. The ego-motion estimation backbones are evaluated on common criteria including average translation error (t_{err}), average rotation error every 100m (r_{err}), Absolute Trajectory Error (ATE), and Relative Pose Error (RPE). The ATE measures the root mean square error between the estimated pose and the ground truth, while the RPE measures frame-by-frame relative pose error.

Table 7.2 summarizes the performance of ORB-SLAM 2 [57] and DF-VO [103] on the set metrics. From the table, it is clear that the DF-VO outperforms the ORB-SLAM 2 in translation drift metrics. On the other hand, it is important to note ORB-SLAM 2 has less rotational drift, even though it has a higher translation

Table 7.2: Quantitative Evaluation of ORB-SLAM 2 [57] and DF-VO [103] on KITTI [37] Odometry Sequences 00–10

Sequence	**ORB-SLAM 2** [57]					**DF-VO** [103]				
	$t_e rr$ (%)	$r_e rr$ (∘/**100m**)	**ATE**	**RPE (m)**	**RPE** (∘)	t_{err} (%)	r_{err} (∘/**100m**)	**ATE**	**RPE (m)**	**RPE** (∘)
00	2.35	0.35	6.03	0.206	0.090	2.33	0.63	14.45	0.039	0.056
01	109.10	0.45	508.34	3.042	0.087	39.46	0.5	117.40	1.554	0.049
02	3.32	0.31	14.76	0.221	0.079	3.24	0.49	19.69	0.057	0.045
03	0.91	0.19	1.02	0.038	0.055	2.21	0.38	1.00	0.029	0.038
04	1.56	0.27	1.57	0.081	0.076	0.74	0.25	0.70	0.026	0.029
05	1.84	0.20	4.04	0.294	0.059	1.30	0.30	4.94	0.018	0.035
06	4.99	0.23	11.16	0.834	0.053	1.42	0.32	3.73	0.025	0.030
07	1.91	0.28	2.19	0.510	0.050	0.72	0.35	1.06	0.015	0.031
08	9.41	0.30	38.85	0.162	0.065	1.66	0.33	6.96	0.030	0.036
09	2.88	0.25	8.39	0.343	0.063	2.07	0.23	7.59	0.044	0.037
10	3.30	0.30	6.63	0.047	0.066	2.06	0.36	4.21	0.040	0.043
Avg	3.24	**0.26**	9.46	0.26	0.06	**1.65**	0.35	**6.09**	**0.02**	**0.03**

drift. This can often be resolved to an extent by employing loop closure techniques, which are inherently supported by ORB-SLAM 2. It is also clear that both methods out perform each other in certain metrics over individual sequences. However, ORB-SLAM 2, being a geometric estimation method, may generalize better on unseen data. Due to this reason, the toolbox provides the user to choose the backbone that works best with their data.

7.3.2 Speed Profile Estimation

The ego vehicle speed profile is mainly used by the trajectory simulation framework to estimate ego vehicle control. Computing accurate speed profiles is a crucial task, without which, the ego vehicle might crash into surrounding road users during simulation. The speed profile ensures that the PID controller estimates the correct control inputs, such that the ego vehicle follows an accurate trajectory. Table 7.3 summarizes the evaluation of speed profile estimation on 8 randomly sampled KITTI tracking sequences. The evaluation takes into account the root mean square error (RMSE) and the absolute error between the ground truth and the estimated speed for the ego vehicle. Figure 7.2 shows the comparison of ground truth and estimated speed profiles for the sequences in table 7.3.

Table 7.3: Quantitative Evaluation of Speed Profile Estimation on 8 KITTI [37] Tracking Sequences

Sequence	**RMSE**	**Absolute Error**
0001	0.304	0.262
0004	0.443	0.420
0011	0.243	0.204
0012	0.004	0.003
0013	0.210	0.197
0014	0.181	0.157
0015	0.184	0.091
0018	0.330	0.292

7.4 3D Vehicle Detection & Tracking

The problem statement of estimating road user trajectories is reformulated as a 3D object detection and tracking problem. The toolbox internally uses quasi-dense 3D detection and tracking [80] for estimating 2D region proposals and 3D bounding box estimates. The toolbox then utilizes the estimated 3D bounding boxes and the ego vehicle motion to estimate a trajectory for each road user in the global frame. Additionally, trajectory smoothing is applied which greatly impacts the 3D tracking performance, beating the current SOTA in vision-only methods. The vehicle detection & tracking processing stage is evaluated on two main metric families - (1) 3D detection metrics, and (2) 3D tracking metrics. This section first lays down the evaluation metrics and then summarizes the performance of the vehicle detection & tracking stage on the set metrics.

The performance of 3D detection is evaluated using the formal evaluation metrics of 3D mean Average Precision (mAP) from KITTI and COCO. mAP in the context of object detection is generally estimated by computing the Average Precision (AP) over different thresholds of Intersection-over-Union (IoU), as shown in equation 7.2. AP is a metric that represents the weighted sum of precision values at each threshold where the weight is the increase in corresponding recall and is computed using equation 7.1.

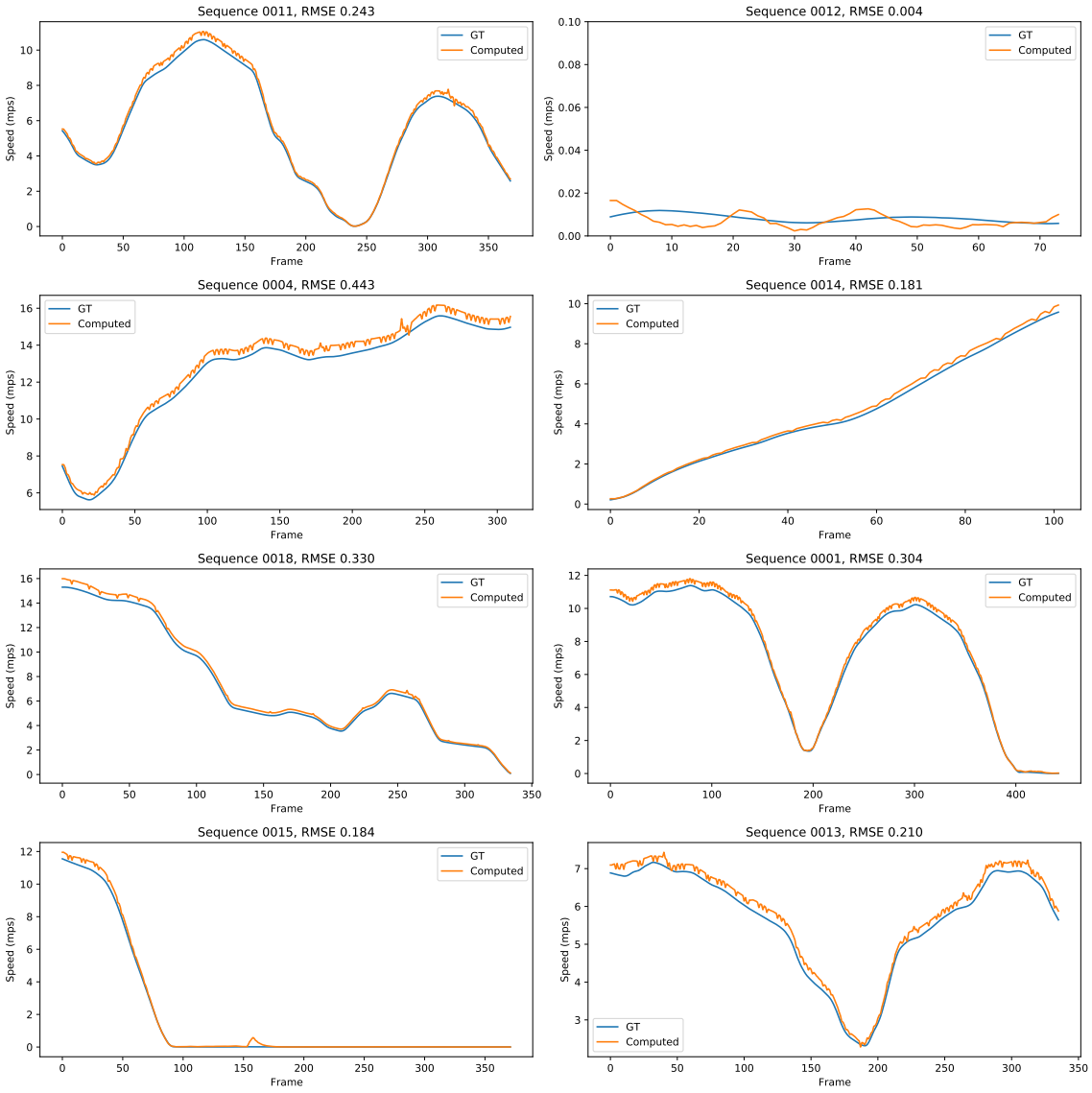


Figure 7.2: Comparison of Estimated and Ground Truth Speed Profiles on 8 KITTI Tracking Sequences

IoU is computed by estimating the intersection and union areas between the predicted and the ground truth values and then taking a ratio of the same. The higher the IoU, the better the prediction.

$$AP = \sum_{i=0}^{i=n-1} (\text{recalls}_i - \text{recalls}_{i+1}) \times \text{precisions}_i \tag{7.1}$$

$$mAP = \frac{1}{n} \sum_{i=0}^{i=n} AP_i \tag{7.2}$$

The performance of 3D tracking is evaluated using official KITTI tracking benchmark metrics. These metrics include Multiple Object Tracking Accuracy ($MOTA$), Multiple Object Tracking Precision (MOTP), Mostly-Tracked (MT), and Mostly Lost (ML). According to CLEAR [106], 3 errors should be taken into account when evaluating a tracker. They can be described as:

1. Miss Detection (FN): This type of error occurs when the object is not detected by the tracker at all.

2. False Positives (FP): This type of error occurs when an object that does not exist is detected by the tracker.

3. Mismatch errors (MM): This type of error occurs when an object is detected, and is present in the ground truth, but switches identity in the tracking lifespan.

$MOTA$ measures the overall accuracy of the tracker by dealing with all three errors as listed above and can be computed as shown in the equation. The toolbox measures the $MOTA_{3D}$ [80], which is denoted as $MOTP_c$, and is based on object centroid distance.

$$MOTA = 1 - \frac{\sum(FN + FP + MM)}{\sum g_t} \tag{7.3}$$

After the 3D bounding box for each road user is computed, the toolbox computes the trajectory of each road user across different frames. Due to the inaccuracies in depth estimation, it is often observed that the trajectory computed using raw 3D bounding box projections is not smooth, and this causes jerky motion during the

trajectory simulation stage. To resolve the jerky motion in simulations, the toolbox applies trajectory smoothing to each individual road user trajectory. Table 7.4 summarizes the evaluation of the performance of two different trajectory smoothing algorithms, (1) Cubic Spline, and (2) Savitzky-Golay filter by re-projecting 3D bounding box locations from the smoothened trajectory. The evaluation in table 7.4 considers three KITTI tracking sequences that are not used for training the underlying quasi-dense detection and tracking framework. From table 7.4 it is clear that trajectory smoothing using Cubic-Spline and Savitzky-Golay filter greatly increases the $MOTA$ performance of the tracker. Further evaluation, as summarized in table 7.5 shows the overall performance of vehicle detection and tracking stage over 8 KITTI tracking sequences that are not used for training, and compares it with the underlying quasi-dense object detection and tracking framework. From the table, it can be noted that the approach presented in this manuscript outperforms the current SOTA [80] by a small margin in the overall results.

7.5 Asset Selection & Scene Setup

Asset selection and scene setup mainly consist of three tasks - (1) CARLA asset selection, (2) Vehicle color detection, and (3) Weather conditions estimation. For the CARLA asset selection stage, the toolbox internally adopts a brute-force approach for CARLA asset selection where-in the estimated dimensions of each road user are compared with the dimensions of all available assets. Table 7.1 summarizes the compute time for the asset selection step for 29 KITTI test sequences. From the table, it can be noted that the toolbox requires 3.15 seconds per road user for asset selection.

The second task of the asset selection and scene setup stage is estimating the vehicle color. There is no ground truth data available for this task, so the toolbox internally uses k-means clustering to estimate the most dominant colors within the

Table 7.4: Quantitative Evaluation of 3D Tracking Stage With Trajectory Smoothing Re-projection on 3 KITTI [107] Sequences

Sequence	**Num frames**	**Range(m)**	**Smoothing**	$MOTA \uparrow$	$MOTP_c \downarrow$	MT	ML
0004	314	0-30	Raw	92.4	**0.63**	0.97	0
			Cubic Spline	95.96	0.64	1	0
			Savitzky-Golay	**96.27**	0.64	1	0
		0-50	Raw	85.5	**0.94**	0.92	0
			Cubic Spline	89.05	0.98	0.93	0.02
			Savitzky-Golay	**89.05**	0.98	0.93	0.02
		0-100	Raw	74.59	**0.98**	0.87	0
			Cubic Spline	76.92	1.03	0.83	0.1
			Savitzky-Golay	**76.92**	1.02	0.78	0.14
0011	373	0-30	Raw	91.22	0.56	0.87	0.02
			Cubic Spline	**92.32**	**0.55**	0.9	0.03
			Savitzky-Golay	92.14	0.55	0.88	0.03
		0-50	Raw	84.6	0.79	0.71	0
			Cubic Spline	**86.54**	0.79	0.71	0.12
			Savitzky-Golay	86.33	**0.78**	0.71	0.1
		0-100	Raw	68.35	0.89	0.6	0.03
			Cubic Spline	**69.03**	**0.89**	0.5	0.34
			Savitzky-Golay	68.97	0.89	0.5	0.32
0015	376	0-30	Raw	84.69	0.5	0.75	0
			Cubic Spline	**96.07**	**0.31**	0.71	0
			Savitzky-Golay	95.45	0.31	0.71	0
		0-50	Raw	82.75	0.63	0.76	0
			Cubic Spline	89.19	0.6	0.75	0.08
			Savitzky-Golay	**89.56**	**0.6**	0.75	0.08
		0-100	Raw	77.06	**0.67**	0.68	0
			Cubic Spline	81.98	0.68	0.72	0.11
			Savitzky-Golay	**82.65**	0.68	0.72	0.11

Table 7.5: Overall Quantitative Evaluation of 3D Tracking Stage and Comparison With SOTA [80]

Range (m)	**Method**	$MOTA \uparrow$	$MOTP_C \downarrow$	MT	ML
0-30	QD-3DT[80]	93.4	0.3	0.95	0
	Ours	**93.44**	**0.29**	0.94	0
0-50	QD-3DT[80]	89.71	0.39	0.9	0
	Ours	**89.74**	**0.38**	0.9	0
0-100	QD-3DT[80]	84.64	0.43	0.88	0
	Ours	**84.83**	**0.42**	0.87	0

bounding box for a road user. After qualitative evaluation and analysis, it was found that the actual color of the car only makes up 10-20% of the overall color composition within each bounding box and that the most dominant color is generally black (due to the windows on the car). Figures 7.3a and 7.3b represent a qualitative evaluation of the car color estimation framework on randomly sampled frames from the KITTI tracking sequences. The figures show the most dominant colors as the color of the bounding box around the vehicle.

(a)

(b)

Figure 7.3: Qualitative Evaluation of Car Color Estimation on Randomly Sampled Frames From KITTI Tracking Sequences

The third task is estimating the weather conditions such that they can be emulated during the trajectory simulation stage. The toolbox internally uses a pre-trained deep CNN that is trained on the BDD100K image tagging dataset [104] using a Deep Layer Aggregation backbone with 34-layers. The deep CNN classifies among 8 classes

including rainy, snowy, clear, overcast, partly cloudy, foggy, and undefined. The table 7.6 summarizes the performance of the deep CNN.

Table 7.6: Quantitative Evaluation of Weather Estimation Deep CNN Using DLA-34 [105] Backbone on BDD-100K [104] Dataset

Method	**Val accuracy**	**Top-1 accuracy**	**Top-5 accuracy**	**Precision**	**Recall**	**F1-score**
DLA-34	81.35	81.35	99.8	79.45	62.67	65.08

7.6 Scenario Replay

This section performs a qualitative evaluation of scenarios generated using a random video found on YouTube[5] and 2 randomly selected KITTI tracking sequences. Figure 7.4 shows the side-by-side comparison between 4 original video frames sampled at different times and their corresponding scenarios generated using the toolbox. Figure 7.5 shows the side-by-side comparison of 4 consecutive original video frames from the KITTI tracking test sequence 04 and the generated simulation in CARLA. This particular example demonstrates a vehicle passing by in the opposite direction. Similarly, Figure 7.6 shows the comparison of 4 consecutive original video frames from the KITTI tracking test sequence 02. This example shows the ability of the trajectory simulation pipeline to sample correct road waypoints and move to the lane that just forked out.

[5] `https://www.youtube.com/watch?v=FHc4od0-if8`

(a) Frame = 113

(b) Frame = 144

(c) Frame = 179

(d) Frame = 257

Figure 7.4: Qualitative Evaluation of Trajectory Simulation a Random Video Obtained From YouTube 5

(a) Frame = 68

(b) Frame = 69

(c) Frame = 70

(d) Frame = 71

Figure 7.5: Qualitative Evaluation of Trajectory Simulation on KITTI [107] Tracking Test Sequence 0004. Left: Frame From Original Video, Right: Corresponding Frame From CARLA

(a) Frame = 146

(b) Frame = 147

(c) Frame = 148

(d) Frame = 149

Figure 7.6: Qualitative Evaluation of Trajectory Simulation on KITTI [107] Tracking Test Sequence 0002. Left: Frame From Original Video, Right: Corresponding Frame From CARLA

Chapter 8

CONCLUSIONS & FUTURE WORK

This thesis introduces a new toolbox named, DeepCrashTest-V2 that is capable of reconstructing high-quality virtual simulations using monocular dashcam videos found on the internet. The toolbox is able to achieve this by employing a multi-stage processing architecture to estimate camera calibration parameters, ego motion, and surrounding road user trajectories on the go. Additionally, the toolbox is able to reconstruct a virtual world in CARLA [15] using the open source map database, OpenStreetMaps [16], where the estimated trajectories are replayed for crucial data collection and analysis. The proposed vehicle detection and tracking framework incorporates the idea of smooth motion for 3D object tracking and outperforms the current state-of-the-art vision-based method on the KITTI [107] multiple object tracking (MOT) benchmark. This thesis also performs the qualitative evaluation of the proposed method on random videos from YouTube to demonstrate its effectiveness.

Even though the proposed toolbox is able to generate usable simulations from user-provided data, there are a few limitations imposed by the adopted methodology. The following limitations can be resolved as part of future enhancements to the toolbox.

1. The toolbox assumes that the data obtained from OpenStreetMaps is accurate and reflects what is seen in the video. This may not always be the case. One way to resolve this issue is to implement a method that creates HD maps on the go.

2. The toolbox assumes that the camera placement is exactly at the center of the

dashboard and neglects any calibration parameters associated with the tilt/pan angle of the camera. This causes the trajectories to be inaccurately estimated when the camera placement is not ideal. Estimating these parameters could greatly improve performance when working with random videos on the internet.

3. The toolbox constructs a basic model of the world by computing road and building geometry from OpenStreetMaps. This can be further extended by the inclusion of GIS and photogrammetry data when available using tools such as CityEngine.

4. The toolbox assumes that the ego vehicle stays on the road throughout the duration of the simulation so that road waypoints can be computed. This causes inaccuracies in generated trajectories when the ego vehicle is involved in a crash and goes off-road.

5. The aspect ratio and resolution of the video greatly affect the calibration parameters. This introduces a problem as most videos found on the internet undergo post-processing, which may cause changes in these properties.

REFERENCES

[1] Todd Litman. Measuring transportation measuring transportation, Mar 2011.

[2] Kenneth A Small and Jose A Gomez-Ibanez. Urban transportation. *Handbook of regional and urban economics*, 3:1937–1999, 1999.

[3] Historical Car Crash Deaths and Rates.

[4] History.com Editors. Automobile history, Apr 2010.

[5] The Need for Automobiles - Montway Auto Transport, May 2011.

[6] Ernst Dieter Dickmanns, Reinhold Behringer, Dirk Dickmanns, Thomas Hildebrandt, Markus Maurer, Frank Thomanek, and Joachim Schiehlen. The seeing passenger car'vamors-p'. In *Proceedings of the Intelligent Vehicles' 94 Symposium*, pages 68–73. IEEE, 1994.

[7] Dean Pomerleau and Todd Jochem. Rapidly adapting machine vision for automated vehicle steering. *IEEE expert*, 11(2):19–27, 1996.

[8] Darsh Parekh, Nishi Poddar, Aakash Rajpurkar, Manisha Chahal, Neeraj Kumar, Gyanendra Prasad Joshi, and Woong Cho. A review on autonomous vehicles: Progress, methods and challenges. *Electronics*, 11(14):2162, 2022.

[9] Karl Iagnemma and Martin Buehler. Editorial for journal of field robotics—special issue on the darpa grand challenge, 2006.

[10] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006.

[11] Darrell Etherington. Over 1,400 self-driving vehicles are now in testing by 80+ companies across the US — techcrunch.com. `https://tcrn.ch/2ZksJCy`. [Accessed 03-Oct-2022].

[12] How Much Automation Does Your Car Really Have? — consumerreports.org. `https://www.consumerreports.org/automotive-technology/how-much-automation-does-your-car-really-have-level-2-a3543419955/`. [Accessed 03-Oct-2022].

[13] SAE International. Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems, j3016_201401, 2014.

[14] Felipe Codevilla, Eder Santana, Antonio M López, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9329–9338, 2019.

[15] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[16] OpenStreetMap Contributors. Planet dump retrieved from https://planet. osm. org. *URL: https://planet. openstreetmap. org*, 2017.

[17] Sai Krishna Bashetty, Heni Ben Amor, and Georgios Fainekos. Deepcrashtest: Turning dashcam videos into virtual crash tests for automated driving systems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11353–11360. IEEE, 2020.

[18] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22(11):1330–1334, 2000.

[19] J-Y Bouguet. Camera calibration toolbox for matlab. *http://www. vision. caltech. edu/bouguetj/calib_doc/index. html*, 2004.

[20] Han Yan, Yu Zhang, Shunli Zhang, Sicong Zhao, and Li Zhang. Focal length estimation guided with object distribution on focalens dataset. *Journal of Electronic Imaging*, 26(3):033018, 2017.

[21] Scott Workman, Connor Greenwell, Menghua Zhai, Ryan Baltenberger, and Nathan Jacobs. Deepfocal: A method for direct focal length estimation. In *2015 IEEE International Conference on Image Processing (ICIP)*, pages 1369–1373. IEEE, 2015.

[22] George Shiu Kai Fung, Nelson Hon Ching Yung, and Grantham KH Pang. Camera calibration from road lane markings. *Optical Engineering*, 42(10):2967–2977, 2003.

[23] Mohammad OA Aqel, Mohammad H Marhaban, M Iqbal Saripan, and Napsiah Bt Ismail. Review of visual odometry: types, approaches, challenges, and applications. *SpringerPlus*, 5(1):1–26, 2016.

[24] Lucas R Agostinho, Nuno M Ricardo, Maria I Pereira, Antoine Hiolle, and Andry M Pinto. A practical survey on visual odometry for autonomous driving in challenging scenarios and conditions. *IEEE Access*, 10:72182–72205, 2022.

[25] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015.

[26] Vikas Thapa, Abhishek Sharma, Beena Gairola, Amit K Mondal, Vindhya Devalla, and Ravi K Patel. A review on visual odometry techniques for mobile robots: Types and challenges. *Recent Advances in Electrical & Electronic Engineering (Formerly Recent Patents on Electrical & Electronic Engineering)*, 13(5):618–631, 2020.

[27] Shashi Poddar, Rahul Kottath, and Vinod Karar. Evolution of visual odometry techniques. *arXiv preprint arXiv:1804.11142*, 2018.

[28] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[29] Yusra Alkendi, Lakmal Seneviratne, and Yahya Zweiri. State of the art in vision-based localization techniques for autonomous navigation systems. *IEEE Access*, 9:76847–76874, 2021.

[30] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *IEEE Robotics & Automation Magazine*, 19(2):78–90, 2012.

[31] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[32] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[33] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[34] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. Ieee, 2011.

[35] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. In *European conference on computer vision*, pages 214–227. Springer, 2012.

[36] Hsiang-Jen Chien, Chen-Chi Chuang, Chia-Yen Chen, and Reinhard Klette. When to use what feature? sift, surf, orb, or a-kaze features for monocular visual odometry. In *2016 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 1–6. IEEE, 2016.

[37] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[38] Sherif AS Mohamed, Mohammad-Hashem Haghbayan, Tomi Westerlund, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. A survey on odometry for autonomous navigation systems. *IEEE Access*, 7:97466–97486, 2019.

[39] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2017.

[40] Nan Yang, Rui Wang, Jorg Stuckler, and Daniel Cremers. Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 817–833, 2018.

[41] Nan Yang, Lukas von Stumberg, Rui Wang, and Daniel Cremers. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1281–1292, 2020.

[42] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2043–2050. IEEE, 2017.

[43] Ran Zhu, Mingkun Yang, Wang Liu, Rujun Song, Bo Yan, and Zhuoling Xiao. Deepavo: Efficient pose refining with feature distilling for deep visual odometry. *Neurocomputing*, 467:22–35, 2022.

[44] Jiahui Huang, Sheng Yang, Tai-Jiang Mu, and Shi-Min Hu. Clustervo: Clustering moving instances and estimating visual odometry for self and surroundings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2168–2177, 2020.

[45] Noha Radwan, Abhinav Valada, and Wolfram Burgard. Vlocnet++: Deep multitask learning for semantic visual localization and odometry. *IEEE Robotics and Automation Letters*, 3(4):4407–4414, 2018.

[46] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 8001–8008, 2019.

[47] Maria Klodt and Andrea Vedaldi. Supervising the new with the old: learning sfm from sfm. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 698–713, 2018.

[48] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5667–5675, 2018.

[49] Chaoyang Wang, José Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2022–2030, 2018.

[50] Lipu Zhou, Jiamin Ye, Montiel Abello, Shengze Wang, and Michael Kaess. Unsupervised learning of monocular depth estimation with bundle adjustment, super-resolution and clip loss. *arXiv preprint arXiv:1812.03368*, 2018.

[51] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3d packing for self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2485–2494, 2020.

[52] Sudeep Pillai, Rareş Ambruş, and Adrien Gaidon. Superdepth: Self-supervised, super-resolved monocular depth estimation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9250–9256. IEEE, 2019.

[53] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[54] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IROS*, volume 3, pages 1442–1447, 1991.

[55] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[56] Iman Abaspur Kazerouni, Luke Fitzgerald, Gerard Dooly, and Daniel Toal. A survey of state-of-the-art on visual slam. *Expert Systems with Applications*, page 117734, 2022.

[57] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[58] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[59] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[60] Shichao Yang and Sebastian Scherer. Cubeslam: Monocular 3-d object slam. *IEEE Transactions on Robotics*, 35(4):925–938, 2019.

[61] Renato F Salas-Moreno, Richard A Newcombe, Hauke Strasdat, Paul HJ Kelly, and Andrew J Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1352–1359, 2013.

[62] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Kosecka. 3d bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7074–7082, 2017.

[63] Buyu Li, Wanli Ouyang, Lu Sheng, Xingyu Zeng, and Xiaogang Wang. Gs3d: An efficient 3d object detection framework for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1019–1028, 2019.

[64] Lijie Liu, Jiwen Lu, Chunjing Xu, Qi Tian, and Jie Zhou. Deep fitting degree scoring network for monocular 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1057–1066, 2019.

[65] Jason Ku, Alex D Pon, and Steven L Waslander. Monocular 3d object detection leveraging accurate proposals and shape reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11867–11876, 2019.

[66] Andretti Naiden, Vlad Paunescu, Gyeongmo Kim, ByeongMoon Jeon, and Marius Leordeanu. Shift r-cnn: Deep monocular 3d object detection with closed-form geometric constraints. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 61–65. IEEE, 2019.

[67] Jiaojiao Fang, Lingtao Zhou, and Guizhong Liu. 3d bounding box estimation for autonomous vehicles by cascaded geometric constraints and depurated 2d detections using 3d results. *arXiv preprint arXiv:1909.01867*, 2019.

[68] Hee Min Choi, Hyoa Kang, and Yoonsuk Hyun. Multi-view reprojection architecture for orientation estimation. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 2357–2366. IEEE, 2019.

[69] Yan Lu, Xinzhu Ma, Lei Yang, Tianzhu Zhang, Yating Liu, Qi Chu, Junjie Yan, and Wanli Ouyang. Geometry uncertainty projection network for monocular 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3111–3121, 2021.

[70] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2147–2156, 2016.

[71] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[72] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[73] Bin Xu and Zhenzhong Chen. Multi-level fusion based 3d object detection from monocular images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2345–2353, 2018.

[74] Hou-Ning Hu, Qi-Zhi Cai, Dequan Wang, Ji Lin, Min Sun, Philipp Krahenbuhl, Trevor Darrell, and Fisher Yu. Joint monocular 3d vehicle detection and tracking. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5390–5399, 2019.

[75] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. *arXiv preprint arXiv:1904.07850*, 2019.

[76] Andrea Simonelli, Samuel Rota Bulo, Lorenzo Porzi, Manuel López-Antequera, and Peter Kontschieder. Disentangling monocular 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1991–1999, 2019.

[77] Eskil Jörgensen, Christopher Zach, and Fredrik Kahl. Monocular 3d object detection and box fitting trained end-to-end using intersection-over-union loss. *arXiv preprint arXiv:1906.08070*, 2019.

[78] Zengyi Qin, Jinglu Wang, and Yan Lu. Triangulation learning network: from monocular to stereo 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7615–7623, 2019.

[79] Garrick Brazil and Xiaoming Liu. M3d-rpn: Monocular 3d region proposal network for object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9287–9296, 2019.

[80] Hou-Ning Hu, Yung-Hsu Yang, Tobias Fischer, Trevor Darrell, Fisher Yu, and Min Sun. Monocular quasi-dense 3d object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[81] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8445–8453, 2019.

[82] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[83] Chen Shangyu. Birdeye - an automatic method for inverse perspective transformation of road image without calibration, Jul 2015.

[84] Bowen Pan, Jiankai Sun, Ho Yin Tiga Leung, Alex Andonian, and Bolei Zhou. Cross-view semantic segmentation for sensing surroundings. *IEEE Robotics and Automation Letters*, 5(3):4867–4873, 2020.

[85] Noureldin Hendy, Cooper Sloan, Feng Tian, Pengfei Duan, Nick Charchut, Yuesong Xie, Chuang Wang, and James Philbin. Fishing net: Future inference of semantic heatmaps in grids. *arXiv preprint arXiv:2006.09917*, 2020.

[86] Chenyang Lu, Marinus Jacobus Gerardus van de Molengraft, and Gijs Dubbelman. Monocular semantic occupancy grid mapping with convolutional variational encoder–decoder networks. *IEEE Robotics and Automation Letters*, 4(2):445–452, 2019.

[87] Samuel Schulter, Menghua Zhai, Nathan Jacobs, and Manmohan Chandraker. Learning to look around objects for top-view representations of outdoor scenes. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 787–802, 2018.

[88] Lennart Reiher, Bastian Lampe, and Lutz Eckstein. A sim2real deep learning approach for the transformation of images from multiple vehicle-mounted cameras to a semantically segmented image in bird's eye view. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–7. IEEE, 2020.

[89] Thomas Roddick and Roberto Cipolla. Predicting semantic map representations from images using pyramid occupancy networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11138–11147, 2020.

[90] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *European Conference on Computer Vision*, pages 194–210. Springer, 2020.

[91] Nikhil Gosala and Abhinav Valada. Bird's-eye-view panoptic segmentation using monocular frontal view images. *IEEE Robotics and Automation Letters*, 7(2):1968–1975, 2022.

[92] Yue Wang, Vitor Campagnolo Guizilini, Tianyuan Zhang, Yilun Wang, Hang Zhao, and Justin Solomon. Detr3d: 3d object detection from multi-view images via 3d-to-2d queries. In *Conference on Robot Learning*, pages 180–191. PMLR, 2022.

[93] Kaustubh Mani, Swapnil Daga, Shubhika Garg, Sai Shankar Narasimhan, Madhava Krishna, and Krishna Murthy Jatavallabhula. Monolayout: Amodal scene layout from a single image. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1689–1697, 2020.

[94] Avishkar Saha, Oscar Mendez, Chris Russell, and Richard Bowden. Translating images into maps. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9200–9206. IEEE, 2022.

[95] Zuoyue Li, Jan Dirk Wegner, and Aurélien Lucchi. Polymapper: Extracting city maps using polygons. *arXiv preprint arXiv:1812.01497*, 2, 2018.

[96] Qi Li, Yue Wang, Yilun Wang, and Hang Zhao. Hdmapnet: A local semantic map learning and evaluation framework. *arXiv preprint arXiv:2107.06307*, 2021.

[97] Avishek Mondal, Panagiotis Tigas, and Yarin Gal. Real2sim: Automatic generation of open street map towns for autonomous driving benchmarks.

[98] Sebastian Maierhofer, Moritz Klischat, and Matthias Althoff. Commonroad scenario designer: An open-source toolbox for map conversion and scenario creation for autonomous vehicles. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3176–3182. IEEE, 2021.

[99] Panpan Cai, Yiyuan Lee, Yuanfu Luo, and David Hsu. Summit: A simulator for urban driving in massive mixed traffic. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4023–4029. IEEE, 2020.

[100] Oleksandr Bogdan, Viktor Eckstein, Francois Rameau, and Jean-Charles Bazin. Deepcalib: A deep learning approach for automatic intrinsic calibration of wide field-of-view cameras. In *Proceedings of the 15th ACM SIGGRAPH European Conference on Visual Media Production*, pages 1–10, 2018.

[101] Jianxiong Xiao, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Recognizing scene viewpoint using panoramic place representation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2695–2702. IEEE, 2012.

[102] Ze Wang, Weiqiang Ren, and Qiang Qiu. Lanenet: Real-time lane detection networks for autonomous driving. *arXiv preprint arXiv:1807.01726*, 2018.

[103] Huangying Zhan, Chamara Saroj Weerasekera, Jia-Wang Bian, Ravi Garg, and Ian Reid. Df-vo: What should be learnt for visual odometry? *arXiv preprint arXiv:2103.00933*, 2021.

[104] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020.

[105] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2403–2412, 2018.

[106] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008:1–10, 2008.

[107] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 963–968. Ieee, 2011.