Real Time Monocular Scene Understanding and Multibody SLAM in Bird's Eye View

Thesis submitted in partial fulfillment
of the requirements for the degree of

*Master of Science in **Electronics and Communication Engineering** by Research*

by

Swapnil Naresh Daga
201531063
swapnil.daga@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
JULY 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Real Time Monocular Scene Understanding and Multibody SLAM in Bird's Eye View" by Swapnil Naresh Daga, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date	Adviser: Prof. K. Madhava Krishna

To my family, friends and teachers

Acknowledgments

First and foremost, I would like to thank my guide Prof. K. Madhava Krishna for giving me an opportunity to be a part of Robotics Research Center, IIIT Hyderabad and advising me throughout the course of all the MS work and the publications to finally this thesis.

I would also like to thank J. Krishna Murthy for being an incredible mentor throughout my work during the MS. I am very much grateful to all my co-authors Gokul B. Nair, Junaid Ahmed Ansari, Rahul Sajnani, Anirudha Ramesh, Kaustubh Mani, N Sai Shankar, Vignesh Prasad, Karmesh Yadav for working constructively together for important submissions.

I would also like to give special thanks to Tayyibah Khanam for all the hardwork and efforts she put in editing, reviewing, researching and suggesting changes to this thesis.

Special Thanks also to Jyotish Poonaganam for all the technical suggestions/alternatives he had and for being available all the time whenever we needed him. I would also like to thank Ritwik Agarwal for helping me with some of the work selflessly despite being from a different research field.

Finally, I would like to thank everyone who I might have missed out unknowingly who were present and supported me during my work in RRC and my stay in IIIT-H.

Abstract

The field of autonomous driving has seen many advances in recent times. In this thesis, we try to look at some important challenges in autonomous driving from the perspective of a single (monocular) camera: Real-Time Scene (Road/Lanes) Layout and multi-object simultaneous localisation and mapping in the outdoor scene. To tackle the challenges faced by monocular setups/systems (such as scale ambiguity in monocular reconstruction, dynamic object localisation, and uncertainty in feature representation), we transpose the above problems into orthographic (*bird's-eye view*) space. Thus, we perform road/lane layout reconstruction, localisation and mapping of dynamic participants and the ego camera vehicle in the scene with an orthographic view as the configuration space. By assuming only the height of the ego-camera above the ground, we leverage single-view metrology cues to accurately localise the ego-vehicle and all other traffic participants in *bird's-eye view*. We demonstrate that our system outperforms prior work that uses strictly greater information, highlighting each design decision's relevance via an ablation analysis.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Related Works . . . 3
1.1.1 3D Object Detection . . . 3
1.1.2 Perspective-View and Depth Estimation . . . 3
1.1.3 Unibody SLAM Frameworks . . . 4
1.1.4 Multibody SLAM Frameworks: . . . 4
1.1.4.1 Traditional Approaches . . . 4
1.1.4.2 Deep Learning-based Approaches . . . 4
1.1.4.3 Recent Approaches . . . 4
1.2 Key Contributions . . . 4
1.3 Thesis Structure . . . 5

2 Background . . . 6
2.1 Lie Groups . . . 6
2.2 SLAM . . . 7
2.2.1 Problem Statement . . . 8
2.2.2 Algorithms . . . 9
2.2.2.1 Filtering Techniques . . . 9
2.2.2.2 Optimization techniques . . . 10
2.3 Pose Graph Optimization . . . 11
2.3.1 Non-Linear Least squares Optimization . . . 12
2.3.2 Sources of Relative Pose Measurement . . . 14
2.4 Morphological Techniques . . . 15
2.4.1 Opening . . . 16
2.4.2 Closing . . . 16
2.4.3 Hough Line Transform . . . 16
2.4.3.1 Hough Space . . . 17
2.4.3.2 Steps involved in line detection . . . 17

3 Real Time Monocular Scene Understanding . . . 20
3.1 Problem statement . . . 20
3.2 Summary of Components . . . 22
3.2.1 Segmentation network . . . 22
3.2.2 Bird's eye view projection . . . 22
3.2.3 Refinement of point clouds . . . 23

3.2.4 Point cloud reconstruction of lanes . . . 24
3.3 Experiments & Results . . . 25
3.3.1 Dataset . . . 25
3.3.2 Qualitative Results . . . 25
3.3.2.1 Real Time Monocular Scene Understanding Pipeline . . . 25
3.3.2.1.1 Input images . . . 25
3.3.2.1.2 Segmented images . . . 25
3.3.2.1.3 Bird's eye view projection . . . 26
3.3.2.1.4 E-Net . . . 27
3.3.2.1.5 Individual lane point clouds . . . 27
3.3.2.2 Extensive E-Net Results . . . 28

4 BirdSLAM . . . 30
4.1 Problem Formulation . . . 30
4.2 Summary of Components . . . 31
4.2.1 *BirdSLAM*: Frontend . . . 31
4.2.1.1 Static Map Initialization . . . 31
4.2.1.2 Scale-Unambiguous Ego-Motion Initialization . . . 32
4.2.1.3 Dynamic Object Localization . . . 32
4.2.2 *BirdSLAM* Backend: Pose-Graph Optimization . . . 32
4.2.2.1 Cost Function . . . 33
4.2.2.2 Constraints . . . 34
4.3 Monocular Scene (Layout) Understanding . . . 35
4.3.1 Generating Amodal Lane Point Clouds in Camera Frame . . . 35
4.4 Experiments and Results . . . 37
4.4.1 Dataset . . . 37
4.4.2 Error Evaluation . . . 37
4.4.3 Approaches Evaluated . . . 38
4.4.3.1 Nair *et al.* [62]: . . . 38
4.4.3.2 CubeSLAM [96]: . . . 38
4.4.3.3 Namdev *et al.* [63]: . . . 38
4.4.3.4 Batch Optimized Baseline in SE(3) with Scale-ambiguous ORB Odometry: . . . 38
4.4.3.5 Incremental Approach in SE(2) with Scale-Initialized Odometry: . . . 39
4.4.4 Qualitative Results . . . 39
4.4.5 Quantitative Results . . . 39
4.4.6 Ablation Studies on Real-Time Approaches . . . 40
4.4.6.1 Contribution by Individual Constraints . . . 40
4.4.6.2 Weight Allotted to Landmark Based Constraints . . . 42
4.4.6.3 Threshold for Landmarks . . . 42
4.4.6.4 Impact of Lane Constraints . . . 42
4.4.6.5 Impact of Dynamic Object Detection Method Comparison . . . 42
4.4.6.6 Runtime Analysis . . . 43

5 Conclusions . . . 44

Bibliography . . . 46

List of Figures

Figure Page

1.1 Top-Left: Illustration of ill-posedness of Multibody SLAM. Triangulating a moving object with a moving camera is impossible as the object has moved away by the time the second image is captured. Back-projected rays intersect at highly erroneous locations. While the car moves along the yellow line, many possible trajectories (red, magenta) project to the same locations in the image. Bottom-Left and Right: BirdSLAM operates in an orthographic view overcoming many nuisance factors like hard-to-obtain geometric matches across vehicles, obstructions and occlusions caused by vehicles and the (already ambiguous) scale of reconstruction drifting unexpectedly and rapidly. It also makes optimizations simpler, and thus convenient to be plugged into downstream planners. While triangulation causes larger error margins for Multibody SLAM, BirdSLAM reduces the error margins by minimizing upon non-linear cost functions in two dimensions (ref. 4.2.2.1) while also taking care of scale reconstruction by taking odometric information and priors in metric scale unlike traditional SLAM like ORB where relative scale can lead to ambiguity. 2

2.1 *Classification of major SLAM algorithms* 10

2.2 *State of the art optimization based SLAM systems consist of a front-end and a back-end block.* 11

2.3 *Graph representation of a SLAM problem with landmarks: Circular nodes represent robot poses **'x'** (yellow) and landmarks **'l'** (green). Edges represent three kinds of spatial constraints between the nodes - blue squares for odometric measurements **'u'** between consecutive robot poses, violet squares for constraints observed due to landmarks **'z'** by the robot, and pink squares signifying loop closures.* 12

2.4 *Representation of the pose-graph SLAM problem: Nodes correspond to robot poses **'x'**, while edges correspond to spatial constraints, that is odometry **'u'** in case of contiguous poses and loop closures **'y'** in all other cases. Each edge between poses x_i and x_j is defined with its error function e_{ij} and the information matrix Ω_{ij}. These error terms arise as a result of the difference between the real measurements y_{ij} and the approximated constraints y'_{ij}.* 13

2.5 Effect of the opening operator 17

2.6 Effect of the closing operator 17

2.7 *Mapping of a unique line from the Cartesian space to the Hough space.* 18

2.8 *Transformation of two pointsp_o and p_1 to two lines in the Hough space. The intersection of both lines in The Hough space indicates the line passing through both points [8].* . . 18

3.1 *Pipeline for Bird's eye view point cloud reconstruction of individual lanes from RGB images* 21
3.2 *KITTI tracking sequences dataset: Frame 151, Sequence 18* 26
3.3 *Semantically segmented image of Frame 151, Sequence 18* 26
3.4 *Occupancy grids for frame 151, sequence 18* 27
3.5 *Input and Output to the E-Net architecture for frame 151, sequence 18* 27
3.6 *Individual Lane Point Cloud Reconstruction for a portion of Sequence 18. The particular frame in bird's eye view is for frame 151 in sequence 18.* 28

4.1 Pipeline: The Ego Motion and Static Map Initialization block illustrates the generation of static road points in 3D, in addition to how we obtain camera trajectory in metric scale. The Dynamic Object Localization block illustrates our approach to obtain two independent sources of localization of other dynamic objects in frame in Birds-Eye View (BEV). Detailed explanation, and mathematical representation of these blocks can be found in Sec. 4.2.1.3. The Backend Pose Graph Optimization block uses the results obtained from Ego Motion and Static Map Initialization block , lane pointclouds obtained from Chapter 3 (the static lane pointclouds marked in pink circles in the pose graph diagram) and the Dynamic Object Localization block to create a pose graph as illustrated. In the pose graph formulation, the red, black, green, and blue arrows show the CC, VV, CV and CP constraints as described in Sec. 4.2.2.2. 31
4.2 Vehicle Localisation in bird's-eye view: The left image shows the 3D bounding box output and the right image shows the tight bounding boxes for the cars we obtain in bird's-eye view in camera frame in metric scale from the procedure described in Sec. 4.2.1.3. The camera center for the right image is at (0,0) of XZ plane facing towards positive Z axis. 33
4.3 Row 1: A frame in KITTI with 3 lanes, one of the lanes being occluded due to obstructions. Row 2, Col 1: "Lane Boundary" occupancy grid pointcloud in bird's-eye view. Row 2, Col 2: "Road" occupancy grid pointcloud in bird's-eye view. Row 3, Col 1: Lane Pointcloud output in bird's-eye view without ENet. Row 3, Col 2: Amodal Lane Pointcloud in bird's-eye view obtained after following procedure in Sec.4.3.1. 36
4.4 Qualitative Results: Visualizations of ego (black colored camera) trajectories and car object trajectories(red and blue color) for some of the KITTI sequences, shown along with surrounding lidar points in metric scale. Some of the results were obtained on very challenging sequences like curved trajectories, occluded detections etc. One such snapshot for a time instance is shown on the right for sequence 20 which had big turns in its path and some of the tracked cars were far away and occluded. 37
4.5 Plot illustrating how number of objects in scene do not affect the time-elapsed in our optimization formulation from Sec. 4.2.2 43

List of Tables

Table		Page
2.1	Different Lie Groups	7
2.2	Terminologies	9
3.1	Extensive ENet results	29
4.1	Absolute Translation Error (in meters) for localised vehicles in scene in bird's-eye view map, computed as root-mean-square error across the 2D axes.	38
4.2	Absolute Translation Error (in meters) for ego motion in bird's-eye view map, computed as root-mean-square error across the 2D axes.	39
4.3	Analysis over the contribution of each type of constraint to the final cost.	40
4.4	Performance of the optimiser as a function of weight given to the landmark based constraints relative to the same for ego motion [column 3 - 5]. Performance of the optimiser with respect to the threshold set for the static feature landmarks on their depth from the camera [column 6 - 10].	41
4.5	Impact of lane-based constraints on batch-based approach from Sec. 4.4.3.4.	41
4.6	Dynamic Object Detection Method Comparison for Real-Time Approach	41

Chapter 1

Introduction

The six levels of autonomy are widely talked about in autonomous driving from level 0 (fully manual) to level 5 (fully autonomous) autonomy [21]. As the race to level 5 autonomy for driverless vehicles moves forward, building accurate and reliable perception modules for the surrounding environment of vehicles becomes of paramount importance.

The response of the perception module of a driverless vehicle module depends a lot on the scene surrounding it. Thus, scene understanding is essential for building any such system. The two major components of an outdoor scene as seen in many autonomous driving datasets [37] are - static components (road, lane markings, trees, buildings, pavements etc.) and dynamic components (vehicles, pedestrians etc.). These two components are often interlinked to each other and accurate perception of one leads to a better understanding of the other. For example, to accurately estimate a dynamic vehicle's trajectory during a lane change, we need to accurately know where the lanes are located. Similarly, during an overtaking manoeuvre, one needs to not only know the accurate localisation of the ego camera vehicle but also the accurate localisation of surrounding vehicles that are in visible vicinity.

A majority of industrially-led solutions to perception modules rely on a suite of sensors such as Lidar, GPS, IMUs, radars, cameras or different permutations of such sensors. In this work, we deviate from this paradigm and pose a challenging research question: "*How accurately can we estimate the ego-motion of a driving platform and the state of the world around it, by using only a single (monocular) camera*"? In robotics parlance, this task of estimating the ego-motion of a "robot" and the state of its environment is referred to as simultaneous localization and mapping (SLAM) [27, 14].

A generalization of the SLAM problem—known as *multibody* SLAM—is of interest to us. While a conventional SLAM system only estimates the robot's ego motion and the static scene map by using the stationary features, multibody SLAM additionally estimates every other actor's motion in the scene - hence a generalized system. This is of paramount importance to autonomous driving platforms, as a precise estimation of the states of other actors immensely boosts the performance of downstream tasks, such as collision avoidance and over-taking manoeuvre.

In general, multibody SLAM is *ill-posed* (i.e., does not admit a unique solution family) in moving monocular camera setup (see Fig. 1.1). This is because monocular reconstruction [59, 33, 44, 24] inher-

ently suffers from *scale factor ambiguity*. This makes it near-impossible to recover object trajectories in metric units that can be directly employed in the downstream tasks mentioned earlier. Thus, monocular cameras have so far found far fewer applications in autonomous driving stacks. In this work, we move monocular SLAM systems one step closer to downstream modules.

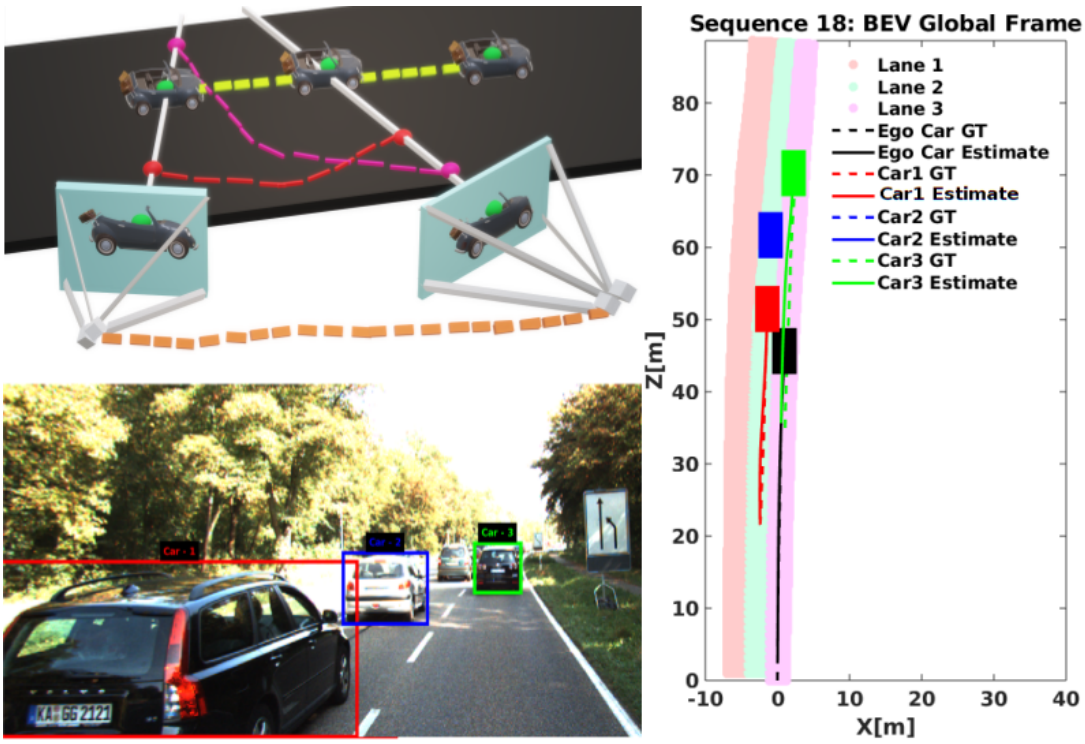


Figure 1.1: Top-Left: Illustration of ill-posedness of Multibody SLAM. Triangulating a moving object with a moving camera is impossible as the object has moved away by the time the second image is captured. Back-projected rays intersect at highly erroneous locations. While the car moves along the yellow line, many possible trajectories (red, magenta) project to the same locations in the image. Bottom-Left and Right: BirdSLAM operates in an orthographic view overcoming many nuisance factors like hard-to-obtain geometric matches across vehicles, obstructions and occlusions caused by vehicles and the (already ambiguous) scale of reconstruction drifting unexpectedly and rapidly. It also makes optimizations simpler, and thus convenient to be plugged into downstream planners. While triangulation causes larger error margins for Multibody SLAM, BirdSLAM reduces the error margins by minimizing upon non-linear cost functions in two dimensions (ref. 4.2.2.1) while also taking care of scale reconstruction by taking odometric information and priors in metric scale unlike traditional SLAM like ORB where relative scale can lead to ambiguity.

Conventional monocular SLAM systems [58, 59] detect and track sparse geometric features across input images and produce a point cloud reconstruction of the scene. These systems are faced with a plethora of issues when deployed in scenes with highly dynamic actors (*e.g.*, traffic): consistent geometric feature matches are hard to obtain across vehicles; the passage of vehicles suddenly obstructs static scene regions with stable features; the (already ambiguous) scale of reconstruction drifts unexpectedly and rapidly. Existing approaches tackle some of these issues by assuming auxiliary inputs such

as optical flow [68] or depth from stereo cameras [69, 50]. Others [23, 90, 41] pose the problem as that of *factorizing* multiple motions from a 3D trajectory "soup". Recent approaches that operate on monocular cameras are unsuitable for real-time applications [62, 96].

In this work, we propose *BirdSLAM*: a monocular multibody SLAM system tailored for typical urban driving scenarios. It operates on an orthographic view (the *bird's-eye view*), where the impact of the aforementioned "nuisance factors" is low also making the optimizations simpler as there are fewer parameters to operate upon. Further, estimates in orthographic views can be directly plugged into downstream planners: a desirable quality (Fig. 1.1). By assuming that all relevant "actions" happen on or close to the ground-plane, and leveraging single-view metrology cues, *BirdSLAM* enables scale-unambiguous motion estimation of the ego vehicle and other traffic participants.

BirdSLAM leverages static features available from an off the shelf SLAM system [59], dynamic features provided by modern object detectors [20, 71, 93], and single-view metrology cues [85, 81] to formulate a scale-aware pose-graph optimization problem in *bird's-eye view*. This can be solved using off-the-shelf pose-graph optimization toolboxes [40, 11, 25]. We demonstrate that *BirdSLAM* outperforms existing full 6-DoF SLAM frameworks and provide an ablation analysis to justify our design choices.

In summary, *BirdSLAM* accurately estimates ego-motion and other vehicle trajectories in *bird's-eye view* over *long sequences* in *real-time*, mitigating the various nuances of traditional 6-DoF SLAM frameworks for dynamic scenes. We observe that a 3-DoF SLAM results on an SE(2) representation of real road plane scenarios compare well with the traditional 6-DoF SLAM results. This simplifies the optimization parameterization thus contributing positively to reduced runtime as shown in Sec. 4.4.6.6 while not sacrificing the Absolute Translation Error(ATE).

1.1 Related Works

1.1.1 3D Object Detection

Various accurate 3D object detection approaches exist using lidar point cloud information [46, 67] and stereo camera depth estimation [49]. However, getting 3D bounding boxes from a monocular camera (single image) is challenging due to the lack of depth information. Recent monocular approaches like OFT [71] and Mono3D [20] solve this problem. Wang *et al.* [93] approaches the above problem for the monocular/stereo case by creating lidar-like representation from monocular/stereo depth maps [38, 19] and then running lidar-based detection networks [46, 67] to get 3D bounding boxes around the objects.

1.1.2 Perspective-View and Depth Estimation

Due to scale ambiguity in monocular reconstruction, estimating depth becomes of paramount importance if we want to lift vehicle trajectories to perspective view (*bird's-eye view*). Various depth

estimation networks like MonoDepth2 [38] attempt to solve this problem. Geometrical methods as described in Song *et al.* [81] and Stein *et al.* [85] try to solve the problem of depth estimation to ground plane from corresponding image pixels by using known priors like the height of the mounted camera, camera intrinsics and normal to the road plane with the assumption of a mostly flat plane.

1.1.3 Unibody SLAM Frameworks

Most popular SLAM systems [59, 33, 36, 92, 44, 24] solve for the unibody (vehicle on which camera is mounted) SLAM problem. However, the estimation of the trajectories of vehicles around becomes essential for autonomous driving.

1.1.4 Multibody SLAM Frameworks:

1.1.4.1 Traditional Approaches

The traditional approaches to solving the SLAM problem's multibody counterpart are based on separating multiple motions [23, 35, 90, 41, 53] from a given set of triangulated points. Other traditional approaches included solving for relative scale for each vehicle in the scene [78, 48, 63]. The relative scale reconstruction in most of such approaches is not in metric scale.

1.1.4.2 Deep Learning-based Approaches

Deep learning-based methods such as Reddy *et al.* [69] and Li *et al.* [50] leverage the improvement in object detection in deep learning approaches over traditional approaches to improve the multibody SLAM. However, these two methods use stereo cameras, thus not facing the problem of scale ambiguity, which is prevalent in monocular settings.

1.1.4.3 Recent Approaches

A more recent approach to the multibody SLAM problem in a monocular setting is proposed by Nair *et al.* [62] which relies on batch-based pose-graph optimization in 6 DoF. The optimization framework used in it cannot be applied in a real-time setting. Another recent framework Cubeslam [96] uses object representations in 6 DoF to improve ego vehicle trajectories; however, the problem is not cast into a dynamic setting, and dynamic participant's trajectories are not shown explicitly.

1.2 Key Contributions

The key contributions of this thesis include the following:

1. A neural network architecture-based pipeline that provides accurate monocular scene understanding in outdoor environments in real-time in *bird's-eye view*.

2. *BirdSLAM*: A Monocular Multibody SLAM framework in *bird's-eye view* that operates in real-time leveraging the above neural network architecture-based pipeline, dynamic and static cues from the environment and a novel pose graph optimization formulation.

1.3 Thesis Structure

The thesis is organized into three main chapters. Chapter 2 provides the necessary context and background required for understanding the major contributions made in this thesis. Chapter 3 focuses on the first contribution - training a neural network architecture-based pipeline for real-time monocular scene understanding. Chapter 4 goes into detail describing *BirdSLAM*, a real-time monocular multibody SLAM in *bird's-eye view* that leverages the pipeline in Chapter 3 along with dynamic vehicle localizations and a novel pose graph formulation.

Chapter 2

Background

The BirdSLAM framework relies on several important concepts that span mathematics, robotics, and computer vision. Understanding these concepts as a whole would provide the reader with sufficient background information as a base before going into the details of BirdSLAM. This section begins with a brief description of Lie groups (Section 2.1), a key mathematical concept for rigid transformations in computer vision problems. Further, a brief introduction of the SLAM problem, its problem statement, and popular techniques for solving SLAM are presented in Section 2.2. Essentially, the pose graph optimization problem plays a key role in the BirdSLAM back-end and hence is described in Section 2.3. Lastly, Morphological techniques come in handy while dealing with segmentation problems and hence Section 2.4 presents some common techniques used in the BirdSLAM framework.

2.1 Lie Groups

The concept of a *group* in mathematics was established to encapsulate the essence of symmetry. A group is thus a collection of symmetries of any object, and every group is a collection of symmetries of some object. Lying at the intersection of algebra and geometry, a Lie group is a group of symmetries where the symmetries are continuous. In other words, a Lie group [29] is a topological group that is also a smooth manifold (e.g.- circle, sphere, etc.), with some other desirable properties. Shapes/objects with a smooth manifold such as the sphere look the same when rotated or displaced by a small amount, thus having continuous symmetries. Due to its ability to provide a robust and coherent framework for working with 3D transformations, the concept of Lie groups is particularly relevant for challenges in robotics and computer vision. These transformations must be composed, differentiated, interpolated, and inverted, something that Lie groups help with [29].

Associated with every Lie group is a Lie algebra, that is the tangent space of Lie groups at the identity. We can build a mapping from the multiplicative structure to an analogous vector space representation by converting Lie groups to Lie algebra [95]. Importantly, a Lie group and its Lie algebra are intimately related, allowing calculations in one to be mapped usefully into the other. Table 2.1 presents some common lie groups with their descriptions.

Group	Description	Dim.	Matrix Representation
SO(3)	3D Rotations	3	3D Rotation Matrix
SE(3)	3D Rigid transformations	6	Linear transformation on homogeneous 4-vectors
SO(2)	2D Rotations	1	2D rotation matrix
SE(2)	2D Rigid Transformations	3	Linear transformation on homogeneous 3-vectors
Sim(3)	3D similarity transformations (rigid motion + scale)	7	Linear transformation on homogeneous 4-vectors

Table 2.1: Different Lie Groups

In this section, our focus relies on two specific Lie group families - SE(3) for 3D rigid transformations and SE(2) for 2D rigid transformations. Subsections below gently brief on the representation of both these lie groups, interested readers are suggested to check [29] for broad descriptions on the lie algebra associated with these lie groups.

- ***SE(3)***: SE(3) has a 6-dimensional manifold, i.e., 6 degrees of freedom - 3 for each 3D translation vector and 3D rotation vector [16]. 6D poses can be be represented by various methods such as 3D+YPR, 3D+Quat and 4×4 transformation matrices. We are concerned mainly with the matrix representation in a 3-dimensional Euclidean space defined by the structure in Equation 2.1, where $R \in SO(3)$ is a proper rotational matrix, $t = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^T \in \mathbb{R}$ is a translation vector and group product the standard matrix product.

$$T = \left(\begin{array}{c|c} R & t \\ \hline 0_{1\times 3} & 1 \end{array} \right) \tag{2.1}$$

- ***SE(2)***: Similar to the SE(3) formulation, a pose (rigid transformation) in 2-dimensional Euclidean space can be determined by a 3×3 homogeneous matrix with the structure shown in equation 2.2 [16]. The SE(2) group has 3 dimensions corresponding to the rotation of ϕ radians and translation *(x, y)* in the 2D plane.

$$T_2 = \left(\begin{array}{c|c} R_2 & t \\ \hline 0_{1\times 2} & 1 \end{array} \right) \tag{2.2}$$

It's worth noting that in both SE(2) and SE(3) formulations discussed above, only *R* and *t* need to be stored in an implementation. The remaining matrix structure can be imposed implicitly [29].

2.2 SLAM

The last two decades have witnessed substantial growth and peaks in the field of robotics and autonomy. The goal was thus to develop cognitive robots capable of serving humans as assistants, or companions and thereby replace humans in many tasks. Since then, researchers all around the world have been paying close attention to the field of mobile robotics and autonomous systems. These two fields of study grew in parallels, with neither having a clear definition until the last several years. A

robot was considered to be a machine capable of making its own decisions, which could then perform a predefined task in a time-efficient and energy-efficient manner. As research progressed, robots were believed to be machines capable of replicating a set of repetitive human tasks [57]. With potential improvements in robot vision, learning, and navigation that occurred throughout time, autonomy was seen to be inextricably linked to the notion of robots, with robots being defined as autonomous devices capable of performing complex tasks to assist humans [83].

To accomplish tasks like navigation and motion, autonomous robots are required to be able to estimate and understand their complicated spatial surroundings as they obtained mobility and manipulation capabilities. Thus emerged the need for autonomous robots to estimate complex environments while in motion, further prompting researchers to develop various localization and mapping techniques. While localization refers to estimating the position of a robot located with respect to its environment, mapping refers to the acquisition of the robot's spatial environment to be able to construct a map. Intuitively, both these techniques rely on one another. Thus, robot localization requires information about the environment in order to determine its position relative to the environment, and subsequently, a robot maps the environment around it based on the robot poses. This necessitated the need for simultaneous localization and mapping, giving rise to the term Simultaneous Localization and Mapping (SLAM). As a result, the solution to the SLAM problem is being regarded as a major breakthrough in autonomous robotics for self-driving car applications.

2.2.1 Problem Statement

In the context of self-driving cars and autonomous vehicles, utilizing the robot's control signals (physical signals) and sensor observation data as inputs, the SLAM problem attempts at constructing a map of the environment using landmarks (stationary features in the environment that are distinguishable) while locating the positions of the robot in the map (robot poses). A mobile robot roaming in an unknown environment collects information through onboard sensors[84]. Since the robot's movement is uncertain, it also faces difficulty in determining its current pose in global coordinates. The SLAM problem then boils down to using this sensor information to acquire a spatial map of the environment while simultaneously localizing the robot's position relative to the map.

Researchers were able to build some early solutions after realizing the necessity for a probabilistic approach to SLAM problems and understanding its consistency to be a fundamental difficulty [28]. A probabilistic approach enables us to explicitly model the uncertainties present due to inaccurate quantities. Hence, solving the SLAM problem requires us to estimate the robot trajectories and the map of the environment, given information on control signals and sensor observations. Equation 2.3 represents a generic SLAM equation, and Table 2.2 briefs about the terms used in this equation.

$$p(x_{0:T}, m | z_{1:T}, u_{1:T}) \tag{2.3}$$

Variable	Description
x_T	Location and Orientation of the robot at time instant '*T*'
z_T	Sensor observations
m	Map and feature locations
u_T	Control signals

Table 2.2: Terminologies

The SLAM problem can be solved either in Online mode or in a Full SLAM fashion. A full SLAM estimates the posterior over the entire path of the robot[45] i.e., estimates the robot path over past sensor information too. On the contrary, online SLAM estimates only the current robot path and the current map of the environment and does not care about previous sensor data. It does so by marginalizing out the previous poses and solving the SLAM problem recursively.Equations 2.4 and 2.5 define online SLAM and full SLAM respectively.

$$p(x_T, m|z_{1:T}, u_{1:T}) \tag{2.4}$$

$$p(x_{1:T}, m|z_{1:T}, u_{1:T}) \tag{2.5}$$

2.2.2 Algorithms

There have been many approaches to solve the SLAM problem, most of which can be categorized into two main paradigms: filtering and optimization based approaches [86]. Filtering algorithms are further classified on the basis of type of filters - Gaussian or Particle filters. Further, graph based techniques constitute a key part of modern state-of-the-art optimization problems and can be classified in three most common graph based approaches. Figure 2.1 illustrates the resulting taxonomy of SLAM, without claiming to be thorough and complete.

2.2.2.1 Filtering Techniques

The problem is modelled as an on-line state estimation when dealing with Bayesian filtering techniques, with the state of the system consisting of the current robot position and the map. Further, the current estimate is refined and augmented utilizing new measurements as they become available [39]. Generally, these filtering based approaches are incremental in nature and hence are also referred to as online SLAM methods[86]. Popular filtering techniques in this category include Particle filters ([55], [56], [32]) and Kalman Filters ([15], [31], [70]), each described below.

- **Extended Kalman Filter (EKF)**: Derived from the original Kalman Filter (KF) used for linear systems, EKF is commonly used for non-linear filtering systems. It introduces a step of linearization for non linear systems, and hence is also applicable to SLAM systems by approximating the mobile robot motion model by means of linear functions [88].

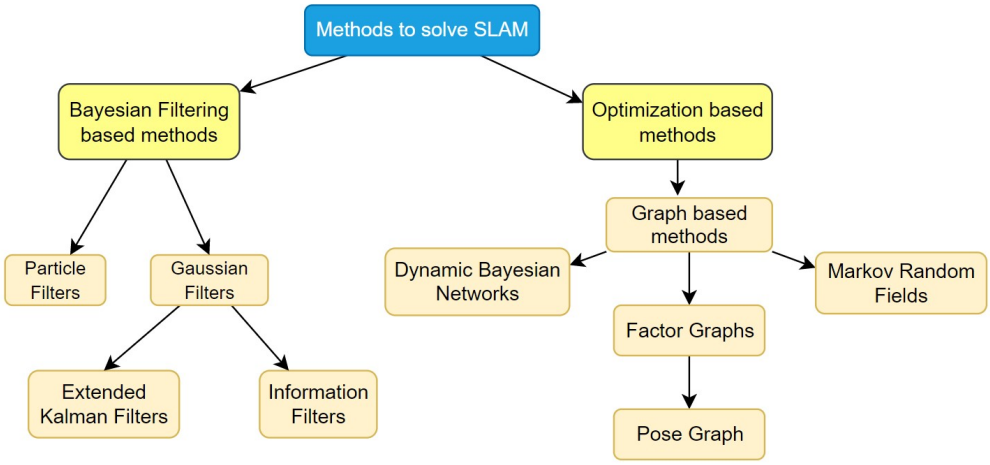


Figure 2.1: *Classification of major SLAM algorithms*

- **Particle Filters**: A set of particles are used in a particle filter to represent a posterior. Here, each particle is a simple guess of the true state value and thus, the particle filter approximates the posterior distribution by collecting many such guesses into a collection of guesses, or set of particles [84].

2.2.2.2 Optimization techniques

Recent research has turned its focus to optimization-based approaches, which have been found to be more efficient, adaptable, scalable, and stable than filtering-based solutions. Optimization based methods to solve the SLAM problem span across linear methods, non-linear least squares methods, relaxation methods, etc. Generally, these optimization-based approaches consist of two parts – the front-end and the back-end. The **front-end** of a SLAM solution is responsible for converting relevant observations acquired in the form of raw sensor data into constraints, while also performing data association tasks. Data association refers to matching sensor observations and associating them with existing map information, i.e., each sensor observation shall be matched with a specific match feature. Additionally, the data association block in the SLAM system consists of a short term block for associating corresponding features in consecutive sensor measurements, as well as a long term block for associating new sensor measurements to older landmarks [18]. The front-end also detects loop closures and hence helps avoid the false positive loop closures [86]. The **back-end** performs inference on the pre-processed data, optimizes them, computes the robot poses and a map of the environment. Usually the back-end also feeds back the information to the front-end to support loop closure detection and validation[18]. Robustness issues due to incorrect data association can also be addressed in the SLAM back-end. When dealing with graph based optimization techniques (discussed below) for solving a SLAM problem, the back-end also relies on the front-end for constructing a topologically correct graph structure, crucial for

convergence. The front-end and back-end blocks of SLAM assembled together form a complete SLAM system, visualized in Figure 2.2.

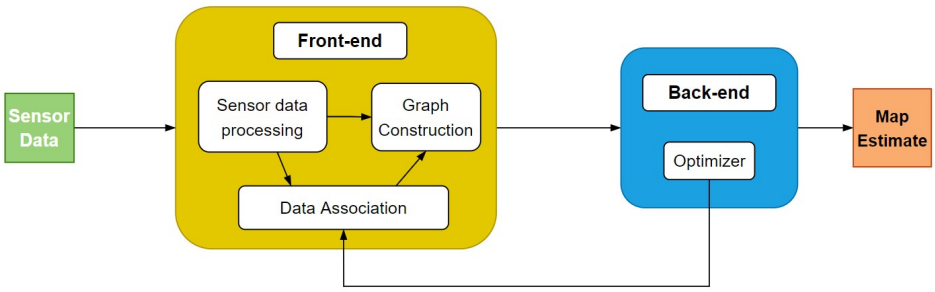


Figure 2.2: *State of the art optimization based SLAM systems consist of a front-end and a back-end block.*

Graph based SLAM: Empirical evaluation and computational tractability being the key drivers of SLAM research have enabled tremendous progress in reducing SLAM into a simpler, practical technology [73]. Further, the SLAM problem can be modelled through different kinds of graph representations such as Dynamic Bayesian Networks (DBNs), Factor Graphs & Markov Random Fields (MRFs). These graph based methods have a sparse structure which makes them computationally efficient and faster [43]. A DBN is a directed cyclic graph where as a factor graph is a bipartite un-directed graph, and a MRF is simply a variant of the factor graph [86]. In this section, we will briefly describe factor graph optimization, also commonly known as graph-SLAM, full smoothing or smoothing and mapping (SAM). Through this formulation, our focus majorly relies on a common graph SLAM variant, also known as the pose-graph SLAM (discussed in the section 2.3). A graph based approach constructs a graph whose nodes represent robot poses or landmarks and an edge between two nodes encodes a sensor measurement that constrains the connected poses and the landmarks with the poses in its first step [39] using raw sensor measurements. These constraints can be highly inaccurate since sensor observations are generally influenced by noise. Figure 2.7 presents a general factor graph representation of the SLAM problem with landmarks. Once a graph has been created, the main challenge is to discover a node configuration that is most consistent with the data through optimization. This necessitates the solution of a large-scale error minimization issue. Thus, the graphical representation of the SLAM problem encouraged researchers to solve the SLAM problem through non-linear sparse optimization.

2.3 Pose Graph Optimization

As compared to occupancy grids or feature maps, pose graphs are more abstract and general maps that have recently gained a lot of interest in SLAM research. Subsequently, a variation to the SLAM problem is the pose-graph SLAM problem, which avoids building an explicit map of the environment

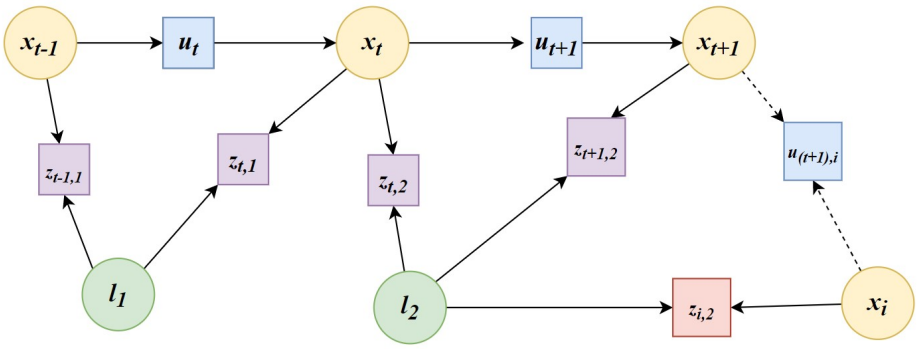


Figure 2.3: *Graph representation of a SLAM problem with landmarks: Circular nodes represent robot poses* ***'x'*** *(yellow) and landmarks* ***'l'*** *(green). Edges represent three kinds of spatial constraints between the nodes - blue squares for odometric measurements* ***'u'*** *between consecutive robot poses, violet squares for constraints observed due to landmarks* ***'z'*** *by the robot, and pink squares signifying loop closures.*

with the objective to estimate the trajectory of the robot given the odometry and loop-closing constraints [43]. In addition to the advantages of graph based SLAM methods, pose slam offers ease while problem solving since it does not build a map of the environment. However, it may fail to converge in case of a significant number of false loop closures since it is not robust against outliers.

In this section, we will specifically focus on the optimization of pose-graph SLAM. While nodes in a pose graph represent robot poses, edges represent spatial constraints between robot poses and can be of two types - odometric constraints and loop closing constraints. Odometric constraints represent edges between continuous robot poses representing information supplied by the robot odometry and the remaining edges representing correspondences between non-continuous robot poses are known as loop closing constraints. Typically, a pose-graph representation of the SLAM problem can be visualized by Figure 2.4. Loop Closing constraints are key to the SLAM problem, because in absence of loop closures, the robot would interpret the outside world as an infinite corridor. Through a loop closure event, a robot informs itself that this corridor keeps intersecting itself, allowing it to better understand the topology of the environment and thus find shortcuts between locations in its path [18]. It's worth noting that explicit landmarks or other environmental parameters aren't included in the pose graph. Such information, on the other hand, can be added to nodes in the graph, allowing the pose graph to describe occupancy grids in 2D or 3D, feature maps, or any combination of those [86]. With this, the goal of pose graph optimization reduced to minimizing the least squared error between all constraints present in the pose graph through an optimal node configuration.

2.3.1 Non-Linear Least squares Optimization

A non linear least squares approach is used to fit a set of m observations with a model that is non-linear in n unknown parameters ($m \geqslant n$) [87]. This method builds on the concept of iterative optimiza-

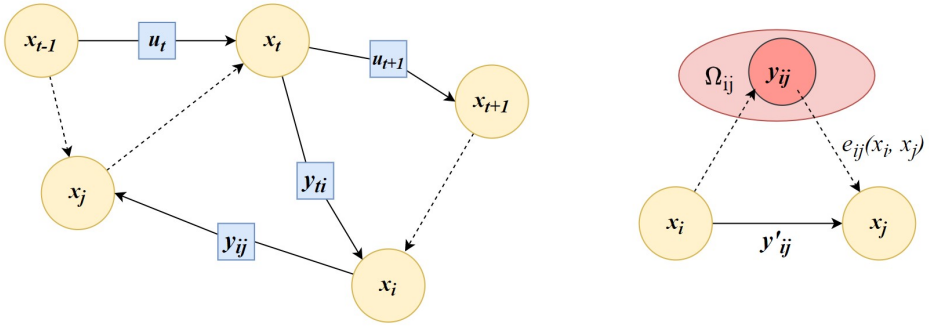


Figure 2.4: *Representation of the pose-graph SLAM problem: Nodes correspond to robot poses* ***'x'****, while edges correspond to spatial constraints, that is odometry* ***'u'*** *in case of contiguous poses and loop closures* ***'y'*** *in all other cases. Each edge between poses x_i and x_j is defined with its error function e_{ij} and the information matrix Ω_{ij}. These error terms arise as a result of the difference between the real measurements y_{ij} and the approximated constraints y'_{ij}.*

tion, where it uses a linear approximation to estimate the model and iteratively modify the parameters. Because an optimization problem is generally defined as finding the minimum of an objective function *F(x)*, we define a non-linear least squares optimization problem in equation 2.6. Here, we are not concerned with the minimum value of the function *F(x)* but with the point x^* where it achieves its minima.

$$x^* = \arg\min_x F(x) \tag{2.6}$$

As mentioned in section 2.2.1, the key to solving the full SLAM problem through a pose-graph representation is estimating the probability function described in equation 2.3. Given a set of odometry and loop closure constraints, the solution seeks the optimal configuration of robot poses denoted by x^*. This optimal solution is also known as *maximum a posteriori* or MAP estimate. The MAP estimate attempts at evaluating the optimal pose configuration x^* where the joint probability distribution P(x—u) has the maximum value, hence seeking the mode of the distribution through the optimization problem [86] visualized in equation 2.7.

$$x^* = \arg\max_x P(x|u) \tag{2.7}$$

Intuitively, both equations 2.6 and 2.7 present the same idea despite of having different formulations. *F(x)* in equation 2.6 can be defined as the sum of errors over all odometric and loop closing constraints in the pose graph as shown in equation 2.8.

$$F(x) = \sum_{\langle i,j\rangle \in C} (e_{ij}^T)(\Omega_{ij})(e_{ij}) \tag{2.8}$$

The collection of index pairings between nodes *i* and *j* is denoted by C, the information matrix between nodes *i* and *j* is represented by Ω_{ij}, e_{ij} is the nonlinear error function that describes how well

the poses x_i and x_j fulfil the constraint imposed by the measurement z_{ij}. Finally, the information matrix Ω_{ij} and the error function e_{ij} are used to model each constraint.

Function *F(x)* is non-linear, that means it can not be expressed as a linear combination of two or more elements and hence requires the application of special iterative approaches such as Gradient descent [77], Gauss Newton [17], Newton's method [34] and Levenberg-Marquardt [76] method. Approximating the error function with its first order Taylor function around the first initial guess is the general idea behind each of these [43]. It is four stepped procedure that starts with fixing an initial guess x_o of the solution. Further, the objective function needs to be approximated as a convex or quasi-convex problem for these iterative approaches to converge towards x^* ([43], [86]). A convex optimization problem is one in which we use iterative computations to identify a point that maximises or minimises the objective function [47]. On solving the convex optimization problem, the found solution x^* may be a local minima of the objective function. In order to find the global minima, these intermediate steps need to be repeated until the point of convergence.

2.3.2 Sources of Relative Pose Measurement

The objective of a pose graph optimization is to estimate robot trajectory from relative pose measurements.The relative pose measurements can be obtained through variety of methods including but not limited to:

- ***Self Motion***: Robotic motion in itself can be captured by on-board sensors such as wheel odometers and Inertial Measurement Units (IMUs). Sensors that measure the rotation of a device's wheels are known as **wheel odometers**. Odometers are simple instruments used for measuring distances travelled by cars, bikes, etc. Distances can be estimated by utilizing the number and speed of wheel rotations, distances between set of two wheels and the difference in rates of both wheel movements in simple calculations and mathematics [80]. **IMU**'s are self contained acceleration and orientation sensing electronic components and are a part of the sensor family. These devices are typically comprised of triads of accelerometers, gyroscopes and sometimes magnetometers [22], enabling 6 to 9 DoFs that is it reports about 6-9 data values. Essentially, this IMU data is related to orientation of the robot, velocity of movement & gravity, important for navigation tasks [10].

- ***Scan Matching***: Data association between two scans is called Scan matching [97]. In order to solve the SLAM problem, it is important to learn rigid body transformations between robot poses and maps - something that is enabled by scan matching algorithms. The scan matching approach efficiently estimates the rigid transformation of the robot between two poses by matching sensor scans acquired from distinct postures. Scanning is a fairly effective way for a mobile robot to locate itself with respect to the given reference scans or maps since exploration sensors are typically quite precise and quick [52]. Thus, the goal of scan matching reduces to finding the best

estimate of a rigid body transformation (translation + rotation) given a scan and a map, or a scan and a scan, or a map and a map [6].

- ***Sparse Feature Registrations***: It is a method of obtaining relative pose measurements or correspondences for accurate visual odometry estimation. Visual odometry plays a crucial role in the context of SLAM since it allows detecting the position and orientation of a robot by examining the associated camera images [64]. In this context, sparse feature registration is a technique that allows building point correspondences between two images based on suitable features. Unlike other methods of image registration that rely on intensity for matching image regions with suitable contrast, sparse feature based registrations register only high structured image regions by establishing feature correspondences in one of the three ways - based on feature dimension & categorization, feature correspondence definition, and on interpolation methods [51].

- ***Dense Point Cloud Reconstruction***: 3D reconstructions from a single image are a hot topic in Computer vision research since they not only allow identification and recognition of objects and shapes, but also recover their full 3D shapes. Existing methods of reconstructions can be divided into four distinct categories - Volumetric , Mesh, Point Cloud and implicit surfaces. In the context of our Bird-SLAM approach, our focus is on point clouds since they are much more memory efficient as compared to other techniques. Point clouds are simply a set of free data points in space representing a 3D object or shape. Point cloud reconstructions are efficient alternatives to voxel-based reconstructions since points are sampled on the surface of an object and thus can effectively capture surface information in depth [54]. The general goal behind point cloud reconstructions is predicting the underlying 3D object as 3D point cloud based on a single input image or a set of input images.

2.4 Morphological Techniques

Thresholding is a common technique in image processing that selects an area of interest from an image, ignoring the parts that we are not concerned with. However, the binary zones created by a basic thresholding operation are distorted and full of noise. Thus, morphological techniques come in hand while dealing with imperfections by taking in account the form and structure of the image.

These techniques are especially suited for binary images (and can be extended for gray-scale images too) since they take into account relative positioning and ordering of pixels instead of their numerical values [9]. In this context, morphological image processing is simply a set of non-linear operations that deal with the shape or morphology of image features. Pre-requisites to understanding morphological techniques also require a brief introduction to the structuring element. A **Structuring Element** is a small binary image window used to analyze an image while applying morphological image processing techniques. The structuring element is placed in all available locations in the image and compared to the pixels in its immediate vicinity [9]. Structuring elements play in morphological image processing the

same role as convolution kernels in linear image filtering. When performing a morphological operation, the origin of the structuring element is normally translated to each pixel position in the image, and the points within the translated structuring element are then compared to the underlying image pixel values [7]. The specifics of this comparison, as well as the impact of the result, are dependent on the morphological operator used.

In this thesis, we will focus mainly on three types of morphological operations relevant to our Bird-SLAM algorithm, namely - Opening, Closing & Hough line transform. Opening & Closing are compound operations, meaning they are both derived from fundamental operations of Erosion [3] and Dilation [2]. Below, we briefly describe Opening and Closing operations in simpler terms, interested readers are recommended to read [42] for derivations and mathematical formulae.

2.4.1 Opening

It has an effect similar to that of erosion, i.e., preserving foreground regions with a similar shape to this structuring element, or that can entirely include the structuring element while removing all other foreground pixels areas [5]. However, the Opening operator is less damaging than erosion. Simply put, an opening is defined as an erosion followed by a dilation, both of which are performed using the same structuring element. As a result, the opening operator needs two inputs: an image to open and a structural element. Since the Opening operator has removed minor specularities and texture fluctuations, the resulting image has a more matt texture than the input image [5].

2.4.2 Closing

It is the dual of the Opening operator (simply opening performed in a reverse fashion), also derived from the two fundamental operators mentioned above. Closing is simply dilation followed by erosion and is similar to the dilation operator as it has the effect of preserving background regions that are comparable in shape to this structuring element or that may entirely contain the structuring element while removing all other background pixels areas [1]. Closing can be used to selectively fill in specific areas of an image's background. It is thus performed in cases where an appropriate structural element can be located that fits well inside areas that should be retained but is not present inside regions that need to be eliminated.

2.4.3 Hough Line Transform

Originally developed and patented by Paul V.C., the Hough transform is used to not only detect complex straight lines in images, but can also detect specific shapes and objects in images. The Hough concept requires understanding of four fundamental concepts - edge detection, mapping edge lines into the Hough space, alternate representations of line, and line detection. Before we describe the four steps

Figure 2.5: Effect of the opening operator

Figure 2.6: Effect of the closing operator

involved in the line detection process using Hough transform, it is important to understand the concept of Hough space.

2.4.3.1 Hough Space

Lines in Cartesian coordinates are typically represented by two parameters, x & y. Equation 2.9 represents the general equation of a line in Cartesian space where a is the slope of the line, and b is its intercept.

$$y = mx + b \tag{2.9}$$

Hough space, on the other hand is characterized by its slope and intercepts, both forming the horizontal and vertical axes respectively. Therefore, the Hough transform uses the form in Equation 2.10, which can be rewritten to Equation 2.11 to be similar to Equation 2.9.

$$r = xcos\theta + ysin\theta \tag{2.10}$$

$$y = -\frac{cos\theta}{sin\theta}x + \frac{r}{sin\theta} \tag{2.11}$$

Therefore, the Hough space has two dimensions and a line can be represented by a single point (r, θ) in the Hough space.

2.4.3.2 Steps involved in line detection

We attempt at providing the readers with a brief overview to the steps involved in the detection of complex straight lines through classical Hough transform.

- **Edge detection**: Edges in an image are defined as regions where the brightness or intensity of pixels changes rapidly. Some common edge detection algorithms include Canny edge detector [26], Sobel operator [91] and the Laplace operator [89]. The output of an edge detector is a

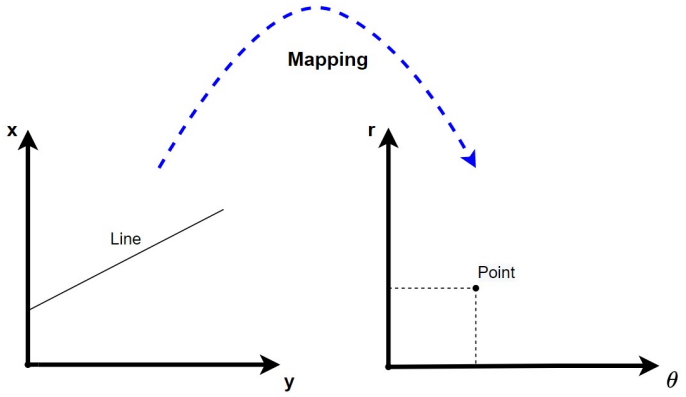
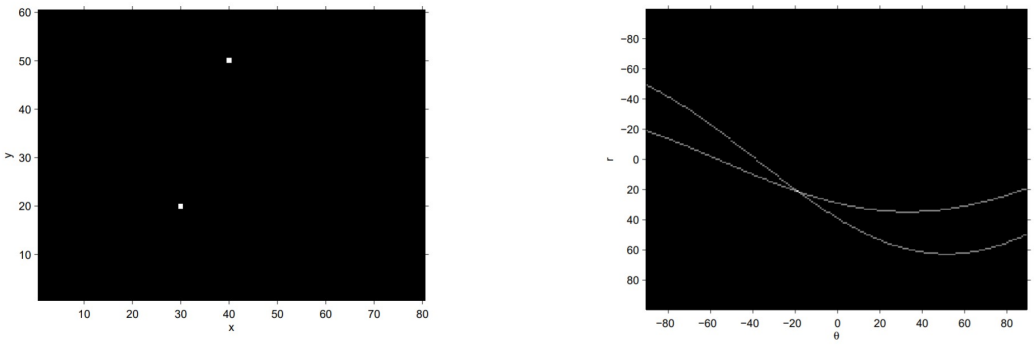


Figure 2.7: *Mapping of a unique line from the Cartesian space to the Hough space.*

binary image, with either 0s or 1s as pixel values also known as an edge image. This edge image is fed as an input to the line detection algorithm.

- **Mapping into the Hough Space**: The Hough transform can be used to identify the parameter(s) of a curve which best fits a set of given edge points (on an edge image). The principle is that all lines that can pass through a point are mapped to that point [8]. One line on the edge image produces a point on the Hough Space since a line is characterized by its slope *a* and intercept *b*.



(a) Two points p_o and p_1

(b) All possible lines through p_o and p_1

Figure 2.8: *Transformation of two pointsp_o and p_1 to two lines in the Hough space. The intersection of both lines in The Hough space indicates the line passing through both points [8].*

- **Interpreting the accumulator**: Once every edge point in the edge map is transformed to all possible lines that could pass through that point, the areas where most Hough space lines intersect is interpreted as true lines in the edge map. Area estimation required in this step is done via an accumulator that covers the entire Hough space.

- **Conversion of infinite lines to finite lines**: Infinite lines are detected by interpretation of the accumulator when all edge points has been transformed. The most basic way the detect lines is to set some threshold for the accumulator, and interpret all values above the threshold as a line. Since the classical Hough transform detects lines by parameters r and θ with no regards about its length, the lines detected after thresholding are infinite lines. Several methods exist for the finite conversion of infinite lines, interested readers are encouraged to explore the same in detail.

Chapter 3

Real Time Monocular Scene Understanding

Scene understanding has become an increasingly crucial task with the growth of robotics and autonomous systems. As a challenging task in computer vision, modern scene understanding systems leverage advances in context modelling, tracking, object detection, and many more [94]. The goal of scene understanding is to give computers human-like vision, to be able to perceive precise location and depth of objects. Monocular vision for visual scene understanding has recently gained popularity by utilizing neural networks for direct depth estimation. Most initial works in monocular depth estimation such as [30], implicitly learn depth cues through supervised learning, by minimizing a regression loss. Further, point clouds come in hand to represent information regarding geometrical context of the image and inter object occlusion in 3D for visual scene understanding. A dense point cloud is thus crucial for tasks such as navigation and planning since it encompasses representation of smaller features and texture details.

Point clouds are directly unattainable from monocular cameras and thus depth maps become critical for obtaining a point cloud from monocular cameras. Depth maps simply contain pixel wise depth information, that can be extrapolated to obtain point clouds in 3D. Thus, this chapter begins by describing the problem statement for real time scene understanding of a busy road in bird's eye view in section. 3.1. Through this problem statement, it introduces the major steps involved in segmentation, 3D reconstruction and refinement of point clouds. Further, sub-components of section 3.2 explain the segmentation architecture, bird's eye view projection, refinement of an occupancy grid, and individual point cloud reconstruction in detail.

3.1 Problem statement

Given a sequence of monocular input images acquired from a camera mounted on top of a moving vehicle on a road with dynamic traffic participants and static road features (for e.g. lane boundaries), the goal is to obtain a 3D point cloud reconstruction of individual lanes in a global ego frame of reference.

1. **Input**: Sequence of monocular RGB images $I = \{I_0, I_2, ...I_{t'}\}$ taken from the front camera of ego vehicle from time $t = 0$ to time $t = t'$.

2. **Output**: Utilizing sequential images from the set I, the goal is to compute individual 3D point clouds as $P = \{P_1, P_2, ...P_n\}$ for each lane in the set of lanes $L = \{L_1, L_2, ...L_n\}$, where n is the number of lanes in the road. Here, each point cloud P_i is made up of singular points p_{1i}, p_{2i}, p_{3i} defined by individual 3D coordinates $\{x, y, z\}$. In definition, every singular point p_{ni} = $\{x_{ni}, y_{ni}, z_{ni}\}$.

Therefore, with the aim of obtaining 3D point cloud reconstructions of individual lanes $P_1, P_2, ...P_n$, the framework includes tasks such as segmentation, bird's eye view projection, refinement, processing, etc. First, Inplace-ABN, a segmentation architecture [74] is used to obtain segmented images of the road and lane boundaries from monocular RGB images in the first step. These segmented images are then projected to Bird's eye view through formula 3.1 [82] to obtain their respective occupancy grids. Feature registrations are then applied for geometric transformations between images in order to compare the information present in images obtained from different measurements. Further, filtering through Opening and Closing operations enables removal of noise and refinement from the registered images. Filtered occupancy grids are then fed as inputs to a pre-trained supervised E-Net architecture [65] to obtain road and lane boundary point clouds in their respective occupancy grids. Finally, several morphological post processing techniques are applied to get individual lane point clouds in the camera frame. Figure 3.1 explains the steps involved in such a point cloud reconstruction from RGB images.

These lane point clouds are used by BirdSLAM in it's SE(2) (Sec. 4.4.3.5) and SE(3) approach (Sec. 4.4.3.4) to fix lateral drifts in the optimization as explained in Sec. 4.2.2.2.

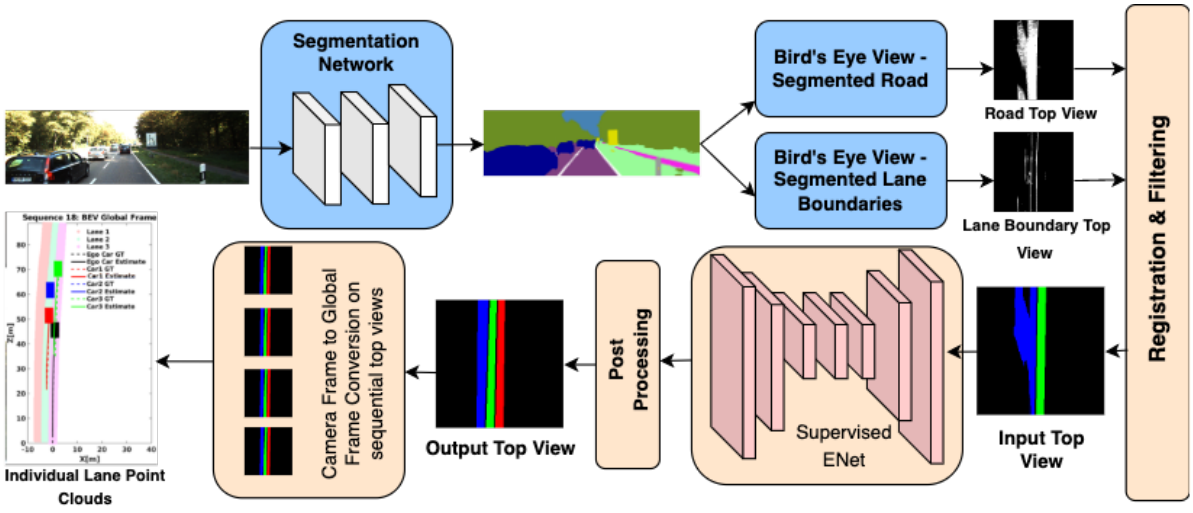


Figure 3.1: *Pipeline for Bird's eye view point cloud reconstruction of individual lanes from RGB images*

3.2 Summary of Components

3.2.1 Segmentation network

Segmentation in computer vision refers to partitioning an image down into two or more sub-groups. Specifically, each pixel in a digital image is labelled as one of the many defined subgroups. Given an input image of an aerial view of a road, the goal of segmentation in this study is to clearly distinguish between road regions and lane boundaries for navigation, planning and control purposes.

Many recent efforts based on modern backbone architectures must limit training batch size to a single batch per GPU, which is partly due to poor memory management in some deep learning frameworks. Obviously, network depth/width significantly correlates with GPU memory needs, and given hardware memory constraints, trade-offs must be made to balance feature extractor speed vs. application-specific characteristics such as network output resolution or training data size. The InPlace-ABN architecture [74] was thus designed to increase the efficiency of training memory for memory demanding and time-consuming tasks such as semantic segmentation. This paper develops a computationally effective solution for checkpointing memory management in the context of Batch Normalization layers, which offers a potential memory throughput of up to 50% and up to 75% on semantic segmentation during training. With images from the image set I, the output of this segmentation architecture are individual segmented images of road and lane boundaries.

3.2.2 Bird's eye view projection

Bird's eye view, also known as aerial view, is observed from an elevated location from those constructed from an imagined bird's perspective. Thus, projection of an image from a front camera view to bird's eye view requires the representation of an image in a shifted coordinate system. Thus, we are interested in the information of each pixel location to relocate it to a new pixel location. This technique of projection to the bird's eye view can be classified under digital image processing as geometrical image modification.

With the segmented images of road and lane boundaries obtained in the previous step, we project these segmented images in the bird's eye view to obtain 2D occupancy grids. Occupancy grids are simply maps of the environment consisting of binary variables that represent the presence of obstacles in the environment. Thus, bird's eye view projection in its initial stage requires information of the location of each object in an image to perform 3D object localization by estimating orientation of the object and ground height. The camera intrinsic calibration matrix K transforms 3D camera coordinates to 2D homogeneous image coordinates. Using this known camera height H, a road point x_p^c in the image space can be back projected in the camera coordinate frame using the matrix K by equation 3.1 first proposed in [82]. Simply, this equation back-projects the bottom of a 2D bounding box x_p^c to 3D through the ground plane $\{H, n\}$.

$$X_p^c = \frac{-HK^{-1}x_p^c}{\bar{n}^T K^{-1} x_p^c} \tag{3.1}$$

3.2.3 Refinement of point clouds

E-Net [65] emerged as a promising architecture for semantic segmentation (visual scene understanding) in real time on low-power mobile devices. Specifically, in this study, this architecture proves to be helpful for refining existing occupancy grids that suffer from data loss due to occlusion and truncation caused by dynamic objects in the scene.

Here, with the inputs as occupancy grids generated in 3.2.2, training is performed on E-Net [65] in a fully supervised fashion on annotated lane point clouds. The output is a refined, smooth and fully formed occupancy grid that does not suffer from any inconsistency and data loss.

While training in a fully supervised fashion, we are faced with a problem of generating and extracting training labels from even occluded sections of the scene since we want a complete scene understanding. Some autonomous driving benchmarks for KITTI provide semantic information along with lidar scans to provide extra information about the scene, however our requirement was for more robust ground truth labels for accurate training. Moreover, lidar information was also not available for all the corresponding road points in a scene.

To generate the ground truth labels for road points, we started with raw lidar data or Monodepth2[38] to initialize our pointclouds corresponding to the road label. The points corresponding to road label were obtained through ground truth semantic segmentation labels wherever available. If ground truth semantic segmentation label was not available, we used state-of-the-art semantic segmentation networks [75] to obtain the same. We got this pointcloud corresponding to road in camera coordinate frame. To overcome a particular frame's occlusion, we aggregated /registered all the pointclouds over a window of 50 frames with the help of odometry information that was available in the KITTI dataset for those particular frames. Few different frames of the same scene over close time intervals cover many of the occluded areas that might not have been covered in a single frame, resulting in a dense noise-free pointcloud of road in that scene. We discarded noisy points greater than a distance of 5m from the car.

The dense pointcloud for roads that we obtained was then projected to an occupancy grid in bird's eye view so that it could be compared with the output from E-Net.

E-Net consists of a fast, compact encoder-decoder architecture that is optimized for fast inference and accuracy. The typical E-Net architecture consists of an initial block, followed by several bottleneck blocks where *conv* is either regular, dilated, or full convolution. Interested readers can refer to the original paper for more details on the architecture [65].

E-Net's superior performance is particularly driven by a few design choices that make it suitable for real time semantic segmentation on low powered devices. Most semantic segmentation architectures including the well-known U-Net architecture [72] downsample images in order to have a bigger receptive field allowing the filter to capture more context. Since downsampling requires equally strong

upsampling, it becomes a computationally expensive task. Of the few approaches that have been proposed to tackle this problem, E-Net relies on saving indices of elements chosen in max pooling layers and using them to produce sparse upsampled maps in the decoder, proposed originally by the authors of SegNet [13]. Further, E-Net's first two blocks significantly reduce the input size using only a small set of feature maps which enables easier and faster processing of input frames, thus optimizing the network in its early stage. However, it is important to understand that rigorous downsampling would hinder information flow in the initial layers of the network. Therefore, an approach similar to the one presented in VGG architectures [79] is utilized. Here, the inference time is increased by 10 times by performing pooling operations parallelly with convolutions, and finally concatenating the resulting feature maps.

Unlike the SegNet architecture consisting of a symmetric encoder-decoder architecture, the authors in ENet proposed to decrease the size of the decoder since its only function is upsampling and fine-tuning. The intuition behind this design choice was also confirmed by experiments replacing all ReLUs in the network with PReLUs, something that had a positive impact on the overall results.

3.2.4 Point cloud reconstruction of lanes

Once the occupancy grids are refined and completed by E-Net in section 3.2.3, a number of morphological post processing techniques including Opening, Closing & Hough line detection are performed. Further, conversion of these occupancy grids to individual lane point clouds requires re-projection in the global frame. A 3D point cloud reconstruction of these 2D occupancy grids in global view essentially requires coordinates of the occupancy grid to first compute a 3D point cloud representation in the local frame. Utilizing a camera transformation matrix $\boldsymbol{T}$, it is then able to compute the point cloud in global ego car's frame. This camera transformation matrix is represented by equation 3.1 where $\boldsymbol{K}$ stands for camera intrinsic parameters, $\boldsymbol{R}$ for rotation of camera in current frame with respect to the starting frame and $\boldsymbol{t}$ for translation of camera in current frame with respect to the starting frame.

$$T = K[R|t] \tag{3.2}$$

Suppose a sequence of images taken from the front camera of an ego vehicle are defined by $I = \{I_1, I_2, ...I_x\}$. In order to reconstruct individual lane point clouds in the global frame, point clouds in the local frame generated using coordinates of the 2D occupancy grid are utilized. with an ego start of I_i till an ego end of I_j where $i \geq 1$ and $j \leq x$, the input is a set of images $I_{ij} = \{I_i, I_{i+1}, ...I_j\}$. A point cloud P'_m for the m^{th} lane in the local frame is converted to the global frame P_m using the transformation matrix. Equation 3.2 describes this conversion within a time duration of $t = \{i, j\}$.

$$P_{m(i,j)} = T * P'_{m(i,j)} \tag{3.3}$$

The local point cloud $P'_{m(i,j)}$ is a concatenation of several point clouds obtained in the defined time duration, i.e., between frames i and j as shown in equation 3.4.

$$P'_{m(i,j)} = P'_{m,i} + P'_{m,i+1} + ... + P'_{m,j} \tag{3.4}$$

Finally, using individual lane points in the local frame over a sequence of images within a defined time duration, we obtain a global point cloud for each lane represented by $P_{m(i,j)}$ where $m = 1, 2, 3...n$. These individual lane point clouds represented by $P_{1(i,j)}, P_{2(i,j)}, P_{3(i,j)}, ..., P_{n(i,j)}$ help in visual understanding of a scene captured between camera frames i and j, essential for tasks in robotics and automation.

3.3 Experiments & Results

3.3.1 Dataset

Our experiments are performed over the KITTI Tracking sequences dataset [4], one of the subsets of the KITTI dataset [37]. KITTI Tracking sequences consists of 21 training sequences and 29 test sequences taken from the front-view camera of a car in motion. Ground truth labels are then obtained from the KITTI Raw dataset [] that contains annotated images that help in supervision, further avoiding the bias.

3.3.2 Qualitative Results

Real Time Monocular Scene Understanding pipeline is a part of the overall BirdSLAM framework that we will be exploring in Chapter 4 as described in the thesis structure. For this reason, we will be showing mainly the Qualitative Results on how the Real Time Monocular Scene Understanding Pipeline works and it's extensive analysis. Quantitative results on how the overall BirdSLAM framework compares to other state-of-the-art SLAM systems is shown in Chapter 4.

3.3.2.1 Real Time Monocular Scene Understanding Pipeline

3.3.2.1.1 Input images The KITTI dataset consists of car driving sequences, where each sequence is made up of a number of individual frames, or RGB images. Each RGB image is fed as an input to our real time monocular scene understanding pipeline. Figure 3.6 contains the RGB image corresponding to frame 151 of the 18^{th} sequence in the KITTI tracking dataset. For demonstrative purposes, each section of this chapter will deal with results corresponding to this particular frame of the said sequence.

3.3.2.1.2 Segmented images The RGB images when fed to the segmentation network as described in 3.2.1 outputs semantically segmented images as shown in figure 3.3. Semantic segmentation refers to the assignment of every pixel of an image corresponding to a particular class label with the same pixel value. Simply, it treats multiple objects of the same class as a single entity. For instance, each object

Figure 3.2: *KITTI tracking sequences dataset: Frame 151, Sequence 18*

corresponding to the class “car” is associated with dark blue pixels, “road” regions are associated with violet pixels, “lane boundaries” with white pixels, “static signboards” with yellow pixels, and so on.

Figure 3.3: *Semantically segmented image of Frame 151, Sequence 18*

3.3.2.1.3 Bird’s eye view projection Consider a dynamic scene with multiple moving cars and stationary road features as in the case of sequences recorded from the front camera of an ego car. Such a scene experiences severe data loss due to occlusions and truncations caused by moving cars. Thus, the real time monocular scene understanding pipeline solves the problem by reconstructing the entire scene of the corresponding frame. It does so by utilizing the segmented images obtained in the previous section. Here, these segmented images are projected to the Bird’s eye view to obtain 2D occupancy grids as discussed in section 3.2.2. Figure 3.4 shows the lane boundary occupancy grids and road occupancy grids in top view.

(a) Lane boundary occupancy grid

(b) Road occupancy grid

Figure 3.4: *Occupancy grids for frame 151, sequence 18*

3.3.2.1.4 E-Net Occupancy grids obtained in the previous section undergo a number of refinement procedures including feature registrations and filtering operations before being fed as an input to the supervised E-Net architecture. Further, the output point cloud of this supervised E-Net undergoes morphological post processing techniques for further refinement and procurement of individual lane point clouds. Figure 3.5 shows the input and output of the E-Net architecture.

(a) Refined input to E-Net

(b) Output of E-Net after post-processing

Figure 3.5: *Input and Output to the E-Net architecture for frame 151, sequence 18*

3.3.2.1.5 Individual lane point clouds As described in section 3.2.4, output of the E-Net is converted to individual lane point clouds in the global frame using the camera transformation matrix and equation 3.2.

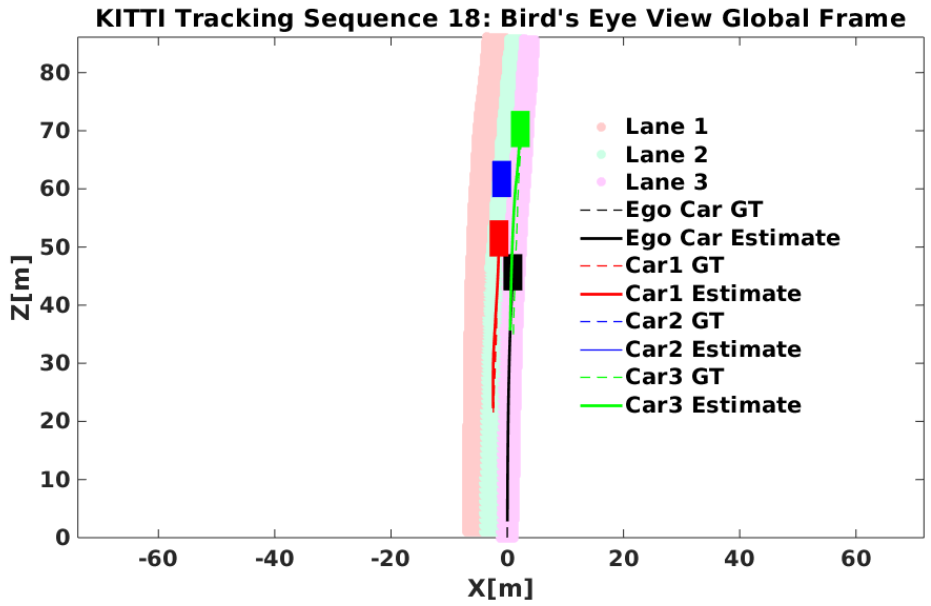


Figure 3.6: *Individual Lane Point Cloud Reconstruction for a portion of Sequence 18. The particular frame in bird's eye view is for frame 151 in sequence 18.*

3.3.2.2 Extensive E-Net Results

Monocular Scene understanding pipeline described in this chapter utilizes the E-Net architecture to obtain refined individual lane point clouds from distorted occupancy grids. Simply, the popular E-Net framework here works by further refinement and completion of occupancy grids obtained after Bird's eye view projection. Table 3.1 shows some of the input RGB images compared with their ENet output lane point clouds. These outputs are obtained after completion and refining by E-Net, before any post-processing is performed.

Experimental results demonstrate the effectiveness of E-net in handling a variety of use cases, from shadows in RGB input images 1, 4, and 5 of Table 3.1 to occlusion and varying light intensities in RGB input image 2, 3, and 6.

S.No	RGB input image	E-Net output
1		
2		
3		
4		
5		
6		

Table 3.1: Extensive ENet results

Chapter 4

BirdSLAM

4.1 Problem Formulation

Given a sequence of monocular images $\mathcal{I}_i, i \in 1 \cdots N$, captured from an urban driving platform, with the camera at height H above the ground, the task of ***BirdSLAM*** is to estimate:

1. The ego motion of the vehicle $X_i = (x_i, z_i, \theta_i)$ at each time step, on the ground plane (assumed to be the XZ plane)

2. An estimate of the motion of all other traffic participants $X_i^j = (x_i^j, z_i^j, \theta_i^j), j \in 1..M_i$, where M_i is the number of traffic participants detected in image I_i.

3. A *map* $\mathcal{M}$ of the environment comprising static features on the road plane (such as lane markings etc.).

The overall pipeline of ***BirdSLAM*** can be seen in Fig. 4.1, where the input images are first passed through an ego motion estimation pipeline (such as an off-the-shelf SLAM system). The resulting estimates are scale-compensated by using single-view metrology cues. In parallel, traffic participants and static scene points on the ground plane are mapped to *bird's-eye view* by a pseudolidar representation [93]. This constitutes the *frontend* of ***BirdSLAM***.

The *backend* of ***BirdSLAM*** comprises a novel multibody pose-graph formulation that employs constraints of several types (CC: camera-camera, CV: camera-vehicle, CP: camera-static map point, VV: vehicle-vehicle) and optimises the pose-graph in real-time to obtain globally-consistent, scale-unambiguous multibody SLAM estimates. In the following subsections, we explain each of these components in detail.

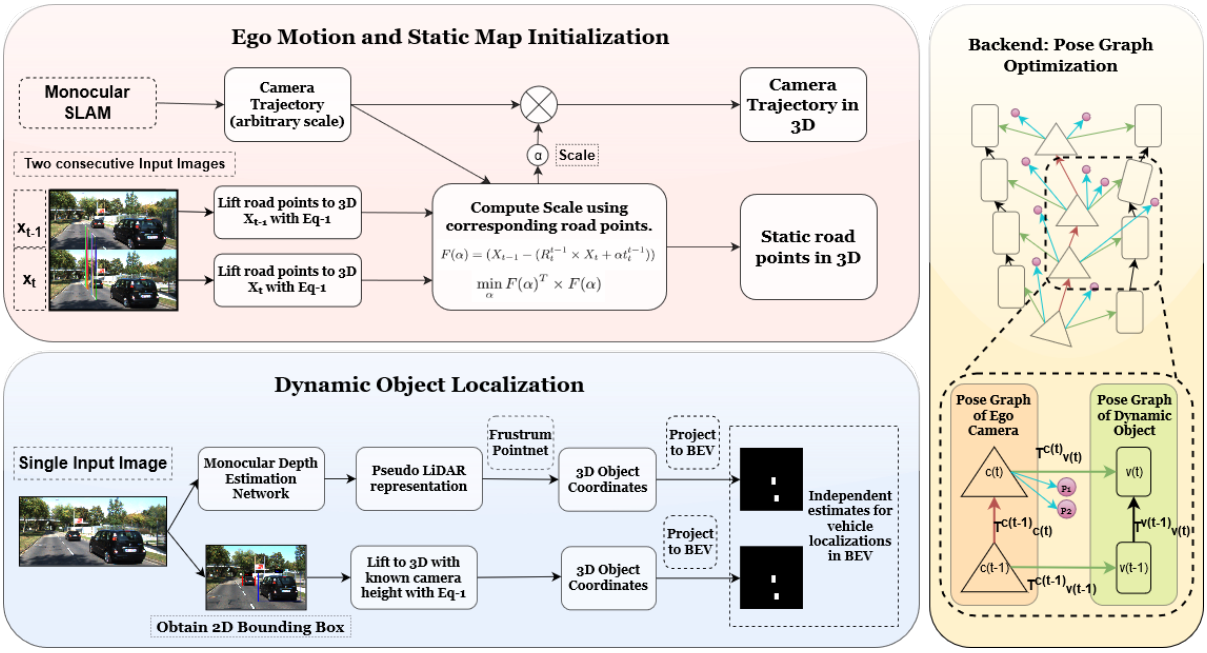


Figure 4.1: Pipeline: The Ego Motion and Static Map Initialization block illustrates the generation of static road points in 3D, in addition to how we obtain camera trajectory in metric scale. The Dynamic Object Localization block illustrates our approach to obtain two independent sources of localization of other dynamic objects in frame in Birds-Eye View (BEV). Detailed explanation, and mathematical representation of these blocks can be found in Sec. 4.2.1.3. The Backend Pose Graph Optimization block uses the results obtained from Ego Motion and Static Map Initialization block , lane pointclouds obtained from Chapter 3 (the static lane pointclouds marked in pink circles in the pose graph diagram) and the Dynamic Object Localization block to create a pose graph as illustrated. In the pose graph formulation, the red, black, green, and blue arrows show the CC, VV, CV and CP constraints as described in Sec. 4.2.2.2.

4.2 Summary of Components

4.2.1 *BirdSLAM*: Frontend

4.2.1.1 Static Map Initialization

Accurately localizing static features in a scene is critical to the success of a feature-based monocular SLAM system. We use ORB features to obtain reliable candidate "stable" features, and prune all those features that do not lie on the road (The "road" region is found by running a lightweight semantic segmentation network [75] over the input image). Using the known camera height H, a road point x_p^c in image space can be back-projected into the camera coordinate frame as follows ($K \in R^{3\times3}$ is the

camera intrinsic matrix, and $\overline{n} \in R^3$ is a unit normal to the ground-plane ($y = 0$))[1]:

$$X_p^c = \frac{-HK^{-1}x_p^c}{\overline{n}^T K^{-1} x_p^c} \tag{4.1}$$

4.2.1.2 Scale-Unambiguous Ego-Motion Initialization

We use ego-motion estimates from an off-the-shelf SLAM system [59] to bootstrap our system. Typically, such estimates are scale-ambiguous. However, upon performing the static map initialization for feature points on road plane using Eqn. 4.1, we obtain map points in metric scale (since the camera height H is known in *meters*; it resolves scale-factor ambiguity). We use a moving-median filter to scale ego-motion estimates to real-world units (typically *meters*). This provides us with a reliant *initialisation for ego motion which is used by the pose-graph* as illustrated in Fig. 4.1 and explained in Sec. 4.2.2 to feed the camera nodes and edges.

4.2.1.3 Dynamic Object Localization

Dynamic traffic participants are the root cause of several monocular SLAM failures. In *BirdSLAM*, we explicitly account (and track!) other vehicles in the scene to provide state estimates that can be directly fed to a downstream planning module. In particular, we employ a monocular depth estimation network [38] and compute a pseudolidar representation [93] using the output depth map. The pseudolidar output is then passed to a Frustum-PointNet [67] to localize vehicles in 3D (see Fig. 4.2). We back project these vehicles localized in 3D to *bird's-eye view* using Eqn. 4.1. We also make use of Eqn. 4.1 on the bottom-center of 2D detection of vehicles in the camera frame as a second unique source of dynamic object localization.

The above module is used to initialize our pose-graph defined in SE(2) in Sec. 4.4.3.5. For initializations to our pose-graph in our baselines in SE(3) in Sec. 4.4.3.4, we make use of the shape-prior based approach [61, 60, 12] for vehicle localizations in the camera's coordinate system.

4.2.2 *BirdSLAM* Backend: Pose-Graph Optimization

We present a lightweight *online* pose-graph formulation that incorporates constraints from multiple entities in the scene (egovehicle, other vehicles, static map features). Each of these constraints contributes a *cost-function* to the optimization process, which we explain below.

[1] *Flat-earth assumption*: For the scope of this paper, we assume that the roads are somewhat planar, i.e., no steep/graded roads on mountains. Consequently, we take normal vector $n = [0, -1, 0]$ in camera frame according to KITTI's [37] conventions where positive x-axis is in right direction, positive y-axis is in downward direction and positive z-axis is in forward direction.

Figure 4.2: Vehicle Localisation in bird's-eye view: The left image shows the 3D bounding box output and the right image shows the tight bounding boxes for the cars we obtain in bird's-eye view in camera frame in metric scale from the procedure described in Sec. 4.2.1.3. The camera center for the right image is at (0,0) of XZ plane facing towards positive Z axis.

4.2.2.1 Cost Function

Following g2o terminologies, the *estimate* $T_S^W \in \mathbb{L}$ characterizes pose for *node* S in global frame W. Here, $\mathbb{L}$ represents the Lie Group in which the respective transformations are defined which could be $SE(2)$ or $SE(3)$. The *measurement* $T_D^S \in \mathbb{L}$ denotes a *binary-edge* from source node S to destination node D effectively constraining the respective *estimates*. This can be represented mathematically as the following transform:

$$\Upsilon_{SD} = (T_D^S)^{-1}(T_S^W)^{-1}(T_D^W) \tag{4.2}$$

We also use *unary-edges* between agent *node* and stationary scene-landmarks p located at $X_p^W \in \mathbb{R}^3$ in the global frame W. Here, the agent A could be ego-camera or the dynamic object in scene. This does not constrain the orientation of the agent. The resultant transform between a agent node A with translation vector $tr_A^W \in \mathbb{R}^3$ and a world landmark p in global frame can be shown as:

$$\Psi_A = tr_A^W - X_p^W \tag{4.3}$$

Our formulation also includes a positive semi-definite inverse covariance matrix or an *information matrix* in each edge's parameterization, shown as $\Omega_E \in \mathbb{R}^{N\times N}$ where $N \in Z$ is the number of degrees of freedom the specific edge E affects. We exploit this to convey confidence of each constraint. We do so by scaling Ω_E upto the *effective information matrix* $\overline{\Omega}_E$ by a factor $\lambda \in \mathbb{R}$ as:

$$\overline{\Omega}_E = \lambda\Omega_E \tag{4.4}$$

From the transforms in Eqn. 4.2 and Eqn. 4.3, we obtain $e^s \in \mathbb{R}^{1\times N}$ by extracting the translation vector directly, and the yaw angle(for $SE(2)$) or the axis-angle rotations(for $SE(3)$). Given the *information matrix* $\Omega^s \in \mathbb{R}^{N\times N}$, we obtain the final cost function for either a *unary* or a *binary-edge* as:

$$\mathbf{F^s} = (e^s)(\Omega^s)(e^s)^T \tag{4.5}$$

4.2.2.2 Constraints

- **Exploiting dynamic cues from vehicles in the scene:** We categorize our pose-graph into three sets of relationships denoting *camera motion*, *vehicle motion* and *camera-vehicle* constraints. Each of these is obtained as a *camera-camera*, *vehicle-vehicle* and *camera-vehicle* edge respectively in consecutive time instants $t-1$ and t. We obtain the final constraint for an m vehicle scenario as:

$$\begin{aligned}\mathcal{F}_D = \mathbf{F}_{C(t-1),C(t)} + \sum_{j=1}^{m} \mathbf{F}^j_{V(t-1),V(t)} \\ + \sum_{j=1}^{m} \mathbf{F}^j_{C(t-1),V(t-1)} + \sum_{j=1}^{m} \mathbf{F}^j_{C(t),V(t)}\end{aligned} \tag{4.6}$$

- **Exploiting static cues using landmarks in the environment:** We also make use of static-cues from the environment to improve agent motion by constraining with respect to the lane. We obtain a dense point-cloud P_l for the road plane segregated for each lane based on Sec. 4.3.1. We define a *unary-edge* between an agent A(Ego-camera or vehicle in scene) and each point p on the lane as shown by Eqn. 4.3.

$$\mathcal{F}_S = \sum_{p} \mathbf{F}_{A,p} \forall (p \in P_l) \tag{4.7}$$

Collectively, the final cost is obtained as the sum of the above Eqn. 4.6 and Eqn. 4.7.

$$\mathcal{F} = \mathcal{F}_D + \mathcal{F}_S \tag{4.8}$$

The scale of the *information matrix* in Eqn. 4.4 is such that higher the scaling(λ), more effective the corresponding cost's *observation* is going to be. Thus, edges with relatively more reliable *observation* are given higher weights while those that bring in higher degrees of error are weighed lower. Thus, CC and CP constraints have the highest weight of 10000 while VV constraints have the lowest of 1. The weight initialization provided to CV constraint ranges between 1000 and 10. The applied weight is gauged according to the depth of the vehicle from the camera. While pseudolidar [93] from Sec. 4.2.1.3 dominates at lower depths, Eqn. 4.1 to 2D vehicle detection bottom-center has an upper hand for far away objects. If the initial ego-motion initialization from the off-the-shelf SLAM systems like ORB

is erroneous, the constraints with the stationary points shown above helps to improve the ego-motion initialization.

4.3 Monocular Scene (Layout) Understanding

4.3.1 Generating Amodal Lane Point Clouds in Camera Frame

The entire process of Generating Amodal Lane Point Clouds is explained in detail in Chapter 3. A brief summary of the pipeline explained in Chapter 3 is given below.

We initialize point clouds in the camera frame using a monocular depth estimation network [38]. Using odometry over a window of W frames, we aggregate sensor observations over time to generate a more dense and noise-free point cloud. To tackle noise in monocular depth estimations, we pick points up to a depth of $5m$ from the camera and then aggregate depths over a larger window size ($\approx 40 - 50$ frames) to compensate for its narrow field of view. This dense point cloud is then projected to an occupancy grid in the *bird's eye view*. We use a state-of-the-art semantic segmentation network [75] to segment each frame and aggregate the *"road"* and *"lane boundary"* prediction point clouds into *separate* occupancy grids (see Fig. 4.3). To achieve more robustness, we apply additional filtering on both of the above occupancy grids by retaining only the patches with more foreground cells than a given threshold in its $m \times m$ neighborhood.

We feed these *"road"* and *"lane boundary"* occupancy grids into ENet [66] to get amodal *"road"* and *"lane boundary"* point clouds in their respective occupancy grids. The above method is especially useful when occlusion from dynamic objects in the scene hinders input data generation. We further apply morphological post-processing techniques like *opening* and *closing*, followed by *hough line transform*. We get segregated amodal lane point clouds in the camera frame in an occupancy grid for each monocular image as shown in Fig. 4.3. These lane point clouds are used by our SE(2) (Sec. 4.4.3.5) and SE(3) approach (Sec. 4.4.3.4) to fix lateral drifts in the optimization as explained in Sec. 4.2.2.2.

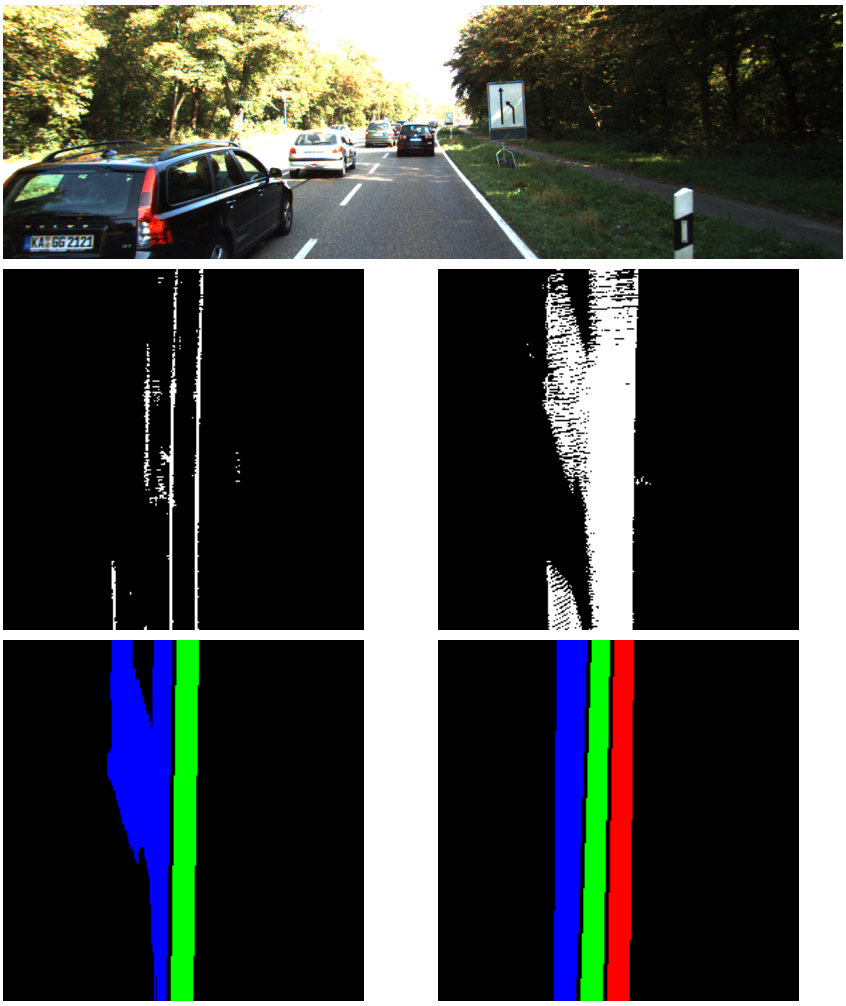


Figure 4.3: Row 1: A frame in KITTI with 3 lanes, one of the lanes being occluded due to obstructions. Row 2, Col 1: ”Lane Boundary” occupancy grid pointcloud in bird’s-eye view. Row 2, Col 2: ”Road” occupancy grid pointcloud in bird’s-eye view. Row 3, Col 1: Lane Pointcloud output in bird’s-eye view without ENet. Row 3, Col 2: Amodal Lane Pointcloud in bird’s-eye view obtained after following procedure in Sec.4.3.1.

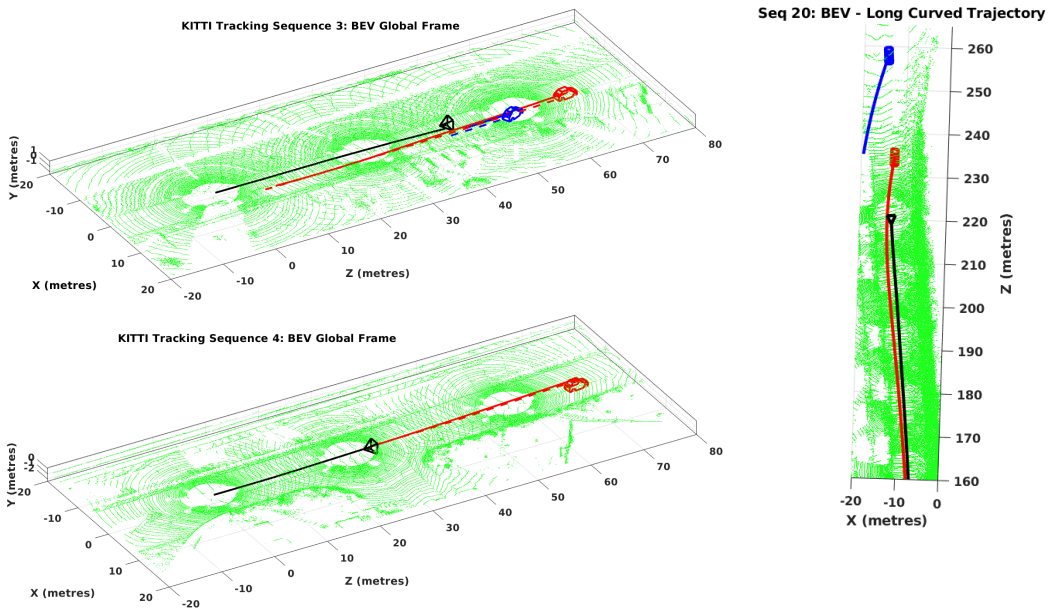


Figure 4.4: Qualitative Results: Visualizations of ego (black colored camera) trajectories and car object trajectories(red and blue color) for some of the KITTI sequences, shown along with surrounding lidar points in metric scale. Some of the results were obtained on very challenging sequences like curved trajectories, occluded detections etc. One such snapshot for a time instance is shown on the right for sequence 20 which had big turns in its path and some of the tracked cars were far away and occluded.

4.4 Experiments and Results

4.4.1 Dataset

We perform experiments over several long KITTI-Tracking sequences [37]. We get ground truth localization to vehicles from the labels available with the dataset and the ground truth ego-motion from the GPS/IMU data given with the dataset.

4.4.2 Error Evaluation

We compute Absolute Translation Error(ATE) as the root-mean-square of error samples for each vehicle's individual frames, including the ego-vehicle in an SE(2) world. Even though the approaches evaluated in Table. 4.1 and Table. 4.2 perform SLAM in SE(3), we project their estimated trajectories onto the ground-plane and compute their error in SE(2) setting for a fair comparison with the results of Sec. 4.4.3.5.

Table 4.1: Absolute Translation Error (in meters) for localised vehicles in scene in bird's-eye view map, computed as root-mean-square error across the 2D axes.

	Absolute Translation Error (RMS) in Global Frame (meters)										
Seq No.	2	3		4	5	10	18			20	
Car ID	1	0	1	2	31	0	1	2	3	12	122
Nair *et al.* [62]	5.01	**1.61**	4.99	2.14	21.64	3.99	1.29	3.45	2.4	9.08	12.86
Namdev *et al.* [63]	6.35	13.81	11.58	11.18	4.09	10.08	3.77	5.93	3.72	25.19	23.76
Ours Sec. 4.4.3.4	6.02	2.20	2.24	**1.77**	**1.76**	3.99	**1.21**	2.86	**1.23**	8.96	13.19
CubeSLAM [96]	–	–	–	–	–	–	1.89	**2.43**	7.17	–	–
Ours Sec. 4.4.3.5	**2.09**	2.37	**2.05**	2.34	1.98	**3.03**	1.6	2.76	1.6	**8.61**	**10.12**

4.4.3 Approaches Evaluated

4.4.3.1 Nair *et al.* [62]:

A monocular multibody approach in SE(3) with a batch-wise pose-graph optimization formulation that resolves relationships with dynamic objects as a means of performing SLAM.

4.4.3.2 CubeSLAM [96]:

A monocular approach that unifies 3D object detections and multi-view object SLAM pipelines in a way that benefits each other.

4.4.3.3 Namdev *et al.* [63]:

A monocular multibody VSLAM approach that obtains motion for dynamic objects and ego-camera in a unified scale. The non tractable relative scale that exists between the moving object and camera trajectories is resolved by imposing the restriction that the object motion is locally linear.

4.4.3.4 Batch Optimized Baseline in SE(3) with Scale-ambiguous ORB Odometry:

A monocular multibody approach in SE(3) similar to Nair *et al.* [62] but the camera nodes are fed with scale-ambiguous ORB [59] initialization. We show that the optimizer itself is able to pull scale-ambiguous odometry to *metric scale* without relying on any prior scale correction like Sec. 4.2.1.2. We incorporate stationary landmarks into this pipeline in the form of a dense lane point cloud for each lane obtained from Sec. 4.3.1 applied in a batch version to correct for the lateral drift contributed to by the relatively erroneous ego-motion initialization.

Table 4.2: Absolute Translation Error (in meters) for ego motion in bird's-eye view map, computed as root-mean-square error across the 2D axes.

	Absolute Translation Error (RMS) in Global Frame (meters)						
Seq No.	2	3	4	5	10	18	20
No. of Frames	67	123	149	101	249	141	414
Nair *et al.* [62]	2.30	1.96	6.49	1.60	10.05	2.40	8.85
Namdev *et al.* [63]	6.24	11.49	11.12	4.08	10.05	3.96	24.38
Ours Sec. 4.4.3.4	**2.05**	1.96	**1.89**	2.22	3.16	2.36	9.05
CubeSLAM [96]	–	–	–	–	–	2.99	–
Ours Sec. 4.4.3.5	2.25	**1.78**	6.60	**1.58**	**2.99**	**1.60**	**8.81**

4.4.3.5 Incremental Approach in SE(2) with Scale-Initialized Odometry:

A variant of the multibody monocular pose-graph based optimization pipeline defined in an SE(2) world. This approach optimizes for multiple objects in each frame in an incremental manner. There is a feedback of "optimization results" back as the input to the optimizer in the next iteration in this approach. The parameterization of the pose-graph optimizer reduces quite considerably in this approach when compared with Sec. 4.4.3.4 as we now function on a world governed by 3 degrees of freedom as opposed to 6.

4.4.4 Qualitative Results

We obtain accurate localizations to vehicles in the camera's view using Sec. 4.2.1.3. The results have been illustrated as tight and accurate 3D bounding boxes obtained to the vehicles in Fig. 4.2. Fig. 4.4 illustrates the trajectories obtained after pose-graph optimization led to accurate bird's-eye view mappings of the ego car and the dynamic vehicles localized in the scene in a stationary world frame. Despite having to cope with a high error contributed to by the motion model predictor from Sec. 4.2.1.2, we obtain close to ground truth *bird's-eye view* mapping post-optimization.

4.4.5 Quantitative Results

Table. 4.2 and Table. 4.1 presents the quantitative performance on a comparative footing for the ego car and the vehicles localized in the camera's scene respectively. Our batch-version with scale-ambiguous odometry initialization showcases a much superior performance compared with other batch-

version, such as Namdev *et al.* [63] and Nair *et al.* [62]. Our batch-approach beats them for all but one vehicle shown in Table. 4.1. On the incremental version's front, we compare our *bird's-eye view* approach with the corresponding baseline defined in SE(3) as well as CubeSLAM [96], whose errors are computed accordingly after running the codebase released by the respective authors on the particular sequence for which the input data and the tuned parameters(for that sequence) were made available. While the performance is comparable between the SE(3) as well as the *bird's-eye view* approach, we put ahead a much superior performance with respect to CubeSLAM [96]. This supports the statement that a *bird's-eye view* SLAM approach can potentially perform as well as its SE(3) counterpart.

Table 4.3: Analysis over the contribution of each type of constraint to the final cost.

Seq No.	Car ID	Absolute Translation Error (RMS) in Global Frame (meters)				
		Without CC	Without CV	Without VV	Without CP	With all
2	1	3.41	1.96	**1.86**	1.95	2.09
	Ego	2.95	2.25	2.25	**2.15**	2.25
3	0	2.35	2.40	2.69	2.46	**2.37**
	1	2.74	2.05	2.23	2.12	**2.05**
	Ego	2.73	1.78	1.78	1.96	**1.78**
4	2	4.52	2.45	**2.26**	2.58	2.34
	Ego	6.82	6.60	6.60	**6.42**	6.60
5	31	3.43	2.01	1.98	1.98	**1.98**
	Ego	3.05	1.58	1.58	**1.57**	1.58
10	0	15.72	**2.81**	2.99	2.98	3.03
	Ego	15.43	2.99	3.52	3.00	**2.99**
18	1	**1.27**	1.65	1.30	1.50	1.60
	2	2.90	2.77	2.84	2.96	**2.76**
	3	1.63	1.82	2.13	1.74	**1.60**
	Ego	2.25	2.21	2.21	2.24	**2.21**
20	12	12.35	8.75	9.33	8.69	**8.61**
	122	17.65	10.32	10.57	**10.09**	10.12
	Ego	13.85	8.86	8.86	8.88	**8.86**

4.4.6 Ablation Studies on Real-Time Approaches

4.4.6.1 Contribution by Individual Constraints

We analyze each constraint's contribution as summarized in Sec. 4.2.2.2 by computing the final error after allotting zero weight to individual constraints, effectively removing its influence on the optimization. The observations are presented in Table. 4.3. Since the CC constraints are given high weight, as explained in Sec. 4.2.2.2, the removal of this constraint results in the deterioration of performance for ego-motion. It can also be seen that, through the CP constraints, the stationary points help enhance ego-motion in most cases. The CV edge that primarily utilizes the pseudolidar [93] based localization ensures that the relation between the ego-motion and all the vehicles in its scene remains synchronized.

Table 4.4: Performance of the optimiser as a function of weight given to the landmark based constraints relative to the same for ego motion [column 3 - 5]. Performance of the optimiser with respect to the threshold set for the static feature landmarks on their depth from the camera [column 6 - 10].

Seq No.	Car ID	Absolute Translation Error (RMS) in Global Frame (meters)							
		Weight to CP			Depth Threshold T (m)				
		Low	Medium	High	12	15	18	20	∞
2	1	2.14	**2.09**	3.15	2.27	2.40	2.14	**2.09**	2.00
	Ego	**2.16**	2.25	2.82	2.36	2.40	2.32	**2.25**	2.28
3	0	2.46	2.37	**2.35**	2.31	2.37	**2.31**	2.37	2.40
	1	2.12	**2.05**	2.76	2.05	2.10	2.14	**2.05**	2.07
	Ego	1.96	**1.78**	2.76	1.80	1.82	1.80	**1.78**	1.87
4	2	**2.17**	2.34	4.67	2.54	2.34	2.36	2.34	**2.26**
	Ego	**6.42**	6.60	5.72	6.59	**6.42**	6.58	6.60	6.45
5	31	1.98	**1.98**	2.05	1.98	1.98	1.98	1.98	**1.90**
	Ego	1.60	**1.58**	1.67	1.58	**1.57**	1.58	1.58	1.60
10	0	**2.99**	3.03	3.30	3.09	3.11	3.03	**3.03**	3.18
	Ego	**2.96**	2.99	3.19	3.01	3.00	2.99	**2.99**	3.07
18	1	**1.50**	1.60	1.28	1.59	**1.59**	1.65	1.60	1.68
	2	2.96	2.76	**2.75**	2.91	2.87	2.83	**2.76**	2.85
	3	1.74	**1.60**	1.80	1.74	1.66	1.62	**1.60**	1.66
	Ego	2.24	2.21	**2.15**	2.45	2.26	2.25	**2.21**	2.25
20	12	8.70	**8.61**	9.38	8.72	8.74	8.68	**8.61**	8.64
	122	**10.09**	10.12	10.36	10.12	**10.10**	10.18	10.12	10.17
	Ego	8.90	**8.86**	9.63	8.88	8.90	**8.84**	8.86	8.85

Table 4.5: Impact of lane-based constraints on batch-based approach from Sec. 4.4.3.4.

	Absolute Translation Error (RMS) in Global Frame (meters)									
Seq No.	3			4		18				Avg Error
Car ID	0	1	Ego-car	2	Ego-car	1	2	3	Ego-car	
Frame length	41	92	123	149	149	62	83	141	141	
Before Lane-Constraints	2.91	2.61	2.26	2.15	4.82	1.32	3.22	**1.19**	2.53	2.56
After Lane-Constraints	**2.20**	**2.24**	**1.96**	**1.77**	**1.89**	**1.21**	**2.86**	1.23	**2.36**	**1.97**

Table 4.6: Dynamic Object Detection Method Comparison for Real-Time Approach

Seq No.	Average Precision (IoU = 0.7) for BEV [Cars] in %		
	Pseudolidar (mono) + Frustum PointNet	Pseudolidar (mono) + AVOD	OFT
2	**23.5**	20	9.5
3	**28.3**	26.8	12.03
4	**34.3**	30.2	15
10	**30**	26	14
18	**26.5**	24.8	11.6

4.4.6.2 Weight Allotted to Landmark Based Constraints

While it has been established from Sec. 4.4.6.1 that static landmarks help improve the absolute translation error of the trajectory, we analyze as to how much emphasis must be given to the CP constraint in terms of the weight. We experiment with various levels of weights fed to the CP constraint in relation with that of the CC edge in the formulation. Table. 4.4 summarizes our observations. While medium weight, which is equal to that of CC constraints, beats other modes by a huge margin in a few instances, it competes closely in all the other instances. On the whole, the performance put forth with medium weight to CP constraints is superior to the other modes.

4.4.6.3 Threshold for Landmarks

Since point correspondences and the depth estimations to the same may be more reliable for features closer to the camera, we place a threshold along Z-axis of the camera to shortlist landmarks to be considered in CP constraints as mentioned in Sec. 4.2.2.2. Our experiments with various thresholds have been reported in Table. 4.4. We find that a threshold $T = 20m$ contributes optimally to the pose-graph optimization step.

4.4.6.4 Impact of Lane Constraints

We show ablation studies on *lane-based constraining* of trajectories in our batch-based pose-graph formulation from Sec. 4.4.3.4. These are performed on unscaled-ORB initializations. We show that lane-constraints contribute by with substantial improvement in ATE for almost all vehicles which are experimented with, when compared with the corresponding ATE before applying lane-based constraints. We summarize our observations in Table. 4.5.

4.4.6.5 Impact of Dynamic Object Detection Method Comparison

We compared different dynamic object detection methods on the same sequence of KITTI datasets to see it's effects in a standalone fashion and on final results. Our observations are summarised in Table 4.6. To compare our results for different object detection methods, we take Average Precision with IoU = 0.7 for all cars in the scene in *bird's-eye view*, which is a standard metric for most of the object detection literature. We compare among three recent state-of-the-art object detection methods in *bird's-eye view* that is pseudolidar with frustum pointnet [93, 67], pseudolidar with AVOD [93, 46]

and OFT [71]. We observe that pseudolidar representation with frustum pointnet outperforms the other object detection methods for the sequences of the KITTI Dataset we test on. This further justifies our decision to use pseudolidar with frustum pointnet in Sec. 4.2.1.3.

4.4.6.6 Runtime Analysis

The incremental optimizer in Sec.4.4.3.5 takes 0.016s to solve the pose-graph optimization problem for a 414 frame long sequence as compared to 1.9s for batch-based approach in Sec.4.4.3.4. Fig. 4.5 shows how a single and multi-object scenario fare in terms of runtime for each incoming instance. Pose-graph optimizations (see 4.2.2) are performed on a quadcore Intel i7-5500U CPU with $2.40GHz$ processor. The frontend involves gathering predictions from multiple neural networks [38, 93, 67] and runs at around 33 Hz frequency.

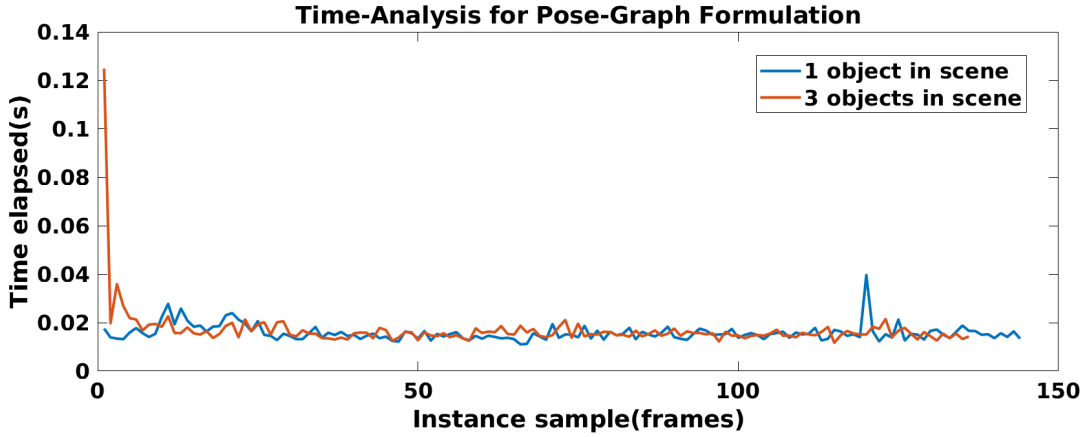


Figure 4.5: Plot illustrating how number of objects in scene do not affect the time-elapsed in our optimization formulation from Sec. 4.2.2

Chapter 5

Conclusions

Multibody SLAM in a moving monocular setup is a difficult problem to solve given its *ill-posedness*. In this paper, we operate in an orthographic (*bird's-eye view*) space to overcome the challenges posed by dynamic scenes to the conventional monocular SLAM systems. Moreover, *BirdSLAM* operates in real-time in *bird's-eye view* space performing better than current real-time state-of-the-art multibody SLAM systems operating in 6 DoF setup. It also performs at par with current offline multibody SLAM systems operating under strictly more resources (time, computation, features). To the best of our knowledge, *BirdSLAM* is the one of the first such system to demonstrate a solution to the multibody monocular SLAM problem in orthographic space. An interesting future direction could be to consider cases in which the single-view metrology cues do not hold, such as on extremely graded/steep roads. Currently, BirdSLAM accounts for the case where ego-motion initialization from off-the-shelf SLAM systems like ORB can be highly erroneous by constraining it with the stationary cues from the environment. Another potentially interesting work could be to improve the fault-tolerance of the BirdSLAM system by taking into account the case when both ego-motion initialization from off-the-shelf SLAM systems as well as stationary points in the environment are highly erroneous.

Related Publications

1. **Swapnil Daga**, Gokul B. Nair, Anirudha Ramesh, Rahul Sajnani, Junaid Ahmed Ansari and K. Madhava Krishna. *BirdSLAM: Monocular Multibody SLAM in Bird's-Eye View*. In **16th International Conference on Computer Vision Theory and Applications (VISAPP)** 2021. Published.

2. (Other) Gokul B. Nair, **Swapnil Daga**, Rahul Sajnani, Anirudha Ramesh, Junaid Ahmed Ansari, Krishna Murthy Jatavallabula, and K. Madhava Krishna. *Multi-Object Monocular SLAM for Dynamic Environments*. In **31st IEEE Intelligent Vehicles Symposium (IV)** 2020. Published.

3. (Other) Kaustubh Mani, **Swapnil Daga**, Shubhika Garg, Sai Shankar Narasimhan, Krishna Murthy Jatavallabhula, K. Madhava Krishna. *MonoLayout: Amodal scene layout from a single image*. In **The IEEE Winter Conference on Applications of Computer Vision (WACV 2020), 1689-1697**. Published.

4. (Other) Vignesh Prasad, Karmesh Yadav, Rohitashva Singh Saurabh, **Swapnil Daga**, Nahas Pareekutty, K Madhava Krishna, Balaraman Ravindran, Brojeshwar Bhowmick. *Learning to Prevent Monocular SLAM Failure using Reinforcement Learning*. In **11th Indian Conference on Computer Vision, Graphics and Image Processing (ICVGIP)** 2020. Published.

Bibliography

[1] Closing. https://homepages.inf.ed.ac.uk/rbf/HIPR2/close.htm.

[2] Dilation. https://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm.

[3] Erosion. https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm.

[4] Object tracking evaluation (2d bounding-boxes).

[5] Opening. https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm.

[6] Scan matching. https://people.eecs.berkeley.edu/~pabbeel/cs287-fa11/slides/scan-matching.pdf.

[7] Structuring elements. https://homepages.inf.ed.ac.uk/rbf/HIPR2/strctel.htm.

[8] Line detection by hough transformation. https://web.ipac.caltech.edu/staff/fmasci/home/astro_refs/HoughTrans_lines_09.pdf, 2009.

[9] Morphological image processing. https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm, 2012.

[10] What is an imu in robotics and hardware? https://blog.studica.com/imu-robotics-hardware, 2019.

[11] S. Agarwal, K. Mierle, and Others. Ceres solver. http://ceres-solver.org.

[12] J. A. Ansari, S. Sharma, A. Majumdar, J. K. Murthy, and K. M. Krishna. The earth ain't flat: Monocular reconstruction of vehicles on steep and graded roads from a moving camera. In *IROS*, 2018.

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

[14] T. Bailey and H. Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE robotics & automation magazine*, 13(3):108–117, 2006.

[15] T. Bailey, J. Nieto, J. Guivant, M. Stevens, and E. Nebot. Consistency of the ekf-slam algorithm. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3562–3568. IEEE, 2006.

[16] J.-L. Blanco. A tutorial on se (3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, 3:6, 2010.

[17] J. V. Burke and M. C. Ferris. A gauss—newton method for convex composite optimization. *Mathematical Programming*, 71(2):179–194, 1995.

[18] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[19] J.-R. Chang and Y.-S. Chen. Pyramid stereo matching network. In *CVPR*, 2018.

[20] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *CVPR*, 2016.

[21] S. O.-R. A. V. S. Committee et al. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. *SAE International: Warrendale, PA, USA*, 2018.

[22] W. T. Conlin. Inertial measurement. *arXiv preprint arXiv:1708.04325*, 2017.

[23] J. Costeira and T. Kanade. A multi-body factorization method for motion analysis. In *ICCV*, 1995.

[24] A. J. Davison, I. D. Reid, N. Molton, and O. Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[25] F. Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[26] L. Ding and A. Goshtasby. On the canny edge detector. *Pattern recognition*, 34(3):721–725, 2001.

[27] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[28] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[29] E. Eade. Lie groups for 2d and 3d transformations. *URL http://ethaneade. com/lie. pdf, revised Dec*, 117:118, 2013.

[30] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *Advances in neural information processing systems*, 27, 2014.

[31] G. A. Einicke and L. B. White. Robust extended kalman filtering. *IEEE transactions on Signal Processing*, 47(9):2596–2599, 1999.

[32] A. I. Eliazar and R. Parr. Dp-slam 2.0. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 2, pages 1314–1320. IEEE, 2004.

[33] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *ECCV*, 2014.

[34] A. Fischer. A special newton-type optimization method. *Optimization*, 24(3-4):269–284, 1992.

[35] A. W. Fitzgibbon and A. Zisserman. Multibody structure and motion: 3-d reconstruction of independently moving objects. In *ECCV*, 2000.

[36] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza. Svo: Semidirect visual odometry for monocular and multicamera systems. *IEEE Transactions on Robotics*, 2017.

[37] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *IJRR*, 2013.

[38] C. Godard, O. Mac Aodha, M. Firman, and G. Brostow. Digging into self-supervised monocular depth estimation. *arXiv preprint*, 2018.

[39] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[40] G. Grisetti, R. Kümmerle, H. Strasdat, and K. Konolige. g2o: a general framework for (hyper) graph optimization. In *ICRA*, 2011.

[41] M. Han and T. Kanade. Multiple motion scene reconstruction from uncalibrated views. In *ICCV*, 2001.

[42] R. M. Haralick, S. R. Sternberg, and X. Zhuang. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*, (4):532–550, 1987.

[43] A. Jurić, F. Kendeš, I. Marković, and I. Petrović. A comparison of graph optimization approaches for pose estimation in slam. In *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1113–1118. IEEE, 2021.

[44] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pages 83–86. IEEE, 2009.

[45] M. Korkmaz, N. Yılmaz, and A. Durdu. Comparison of the slam algorithms: Hangar experiments. In *MATEC Web of Conferences*, volume 42, page 03009. EDP Sciences, 2016.

[46] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.

[47] A. Kumar. Convex optimization explained: Concepts amp; examples, Oct 2021.

[48] A. Kundu, K. M. Krishna, and C. Jawahar. Realtime multibody visual slam with a smoothly moving monocular camera. In *ICCV*, 2011.

[49] P. Li, X. Chen, and S. Shen. Stereo r-cnn based 3d object detection for autonomous driving. In *CVPR*, 2019.

[50] P. Li, T. Qin, et al. Stereo vision-based semantic 3d object and ego-motion tracking for autonomous driving. In *ECCV*, 2018.

[51] C. Lorenz, T. Klinder, and J. v. Berg. Feature-based registration techniques. In *4D Modeling and Estimation of Respiratory Motion for Radiation Therapy*, pages 85–102. Springer, 2013.

[52] J. LV. *Scan Matching and SLAM for Mobile Robot in Indoor Environment*. PhD thesis, Hokkaido University, 2016.

[53] M. Machline, L. Zelnik-Manor, and M. Irani. Multi-body segmentation: Revisiting motion consistency. In *ECCV Workshop on Vision and Modeling of Dynamic Scenes*, 2002.

[54] P. Mandikal and V. B. Radhakrishnan. Dense 3d point cloud reconstruction using a deep pyramid network. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1052–1060. IEEE, 2019.

[55] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[56] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, volume 3, pages 1151–1156, 2003.

[57] H. P. Moravec. Robot.

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

[59] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 2017.

[60] J. K. Murthy, G. S. Krishna, F. Chhaya, and K. M. Krishna. Reconstructing vehicles from a single image: Shape priors for road scene understanding. In *ICRA*, 2017.

[61] J. K. Murthy, S. Sharma, and K. M. Krishna. Shape priors for real-time monocular object localization in dynamic environments. In *IROS*, 2017.

[62] G. B. Nair, S. Daga, R. Sajnani, A. Ramesh, J. A. Ansari, and K. M. Krishna. Multi-object monocular slam for dynamic environments. *arXiv preprint*, 2020.

[63] R. Namdev, K. M. Krishna, and C. V. Jawahar. Multibody vslam with relative scale solution for curvilinear motion reconstruction. In *ICRA*, 2013.

[64] D. Nistér, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. Ieee, 2004.

[65] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.

[66] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint*, 2016.

[67] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018.

[68] R. Ranftl, V. Vineet, Q. Chen, and V. Koltun. Dense monocular depth estimation in complex dynamic scenes. In *CVPR*, 2016.

[69] N. D. Reddy, I. Abbasnejad, S. Reddy, A. K. Mondal, and V. Devalla. Incremental real-time multibody vslam with trajectory optimization using stereo camera. In *IROS*, 2016.

[70] M. I. Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics*, 43:46, 2004.

[71] T. Roddick, A. Kendall, and R. Cipolla. Orthographic feature transform for monocular 3d object detection. *British Machine Vision Conference*, 2019.

[72] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[73] D. M. Rosen, K. J. Doherty, A. Terán Espinoza, and J. J. Leonard. Advances in inference and representation for simultaneous localization and mapping. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:215–242, 2021.

[74] S. Rota Bulò, L. Porzi, and P. Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[75] S. Rota Bulò, L. Porzi, and P. Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *CVPR*, 2018.

[76] S. Roweis. Levenberg-marquardt optimization. *Notes, University Of Toronto*, 1996.

[77] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[78] K. Schindler and D. Suter. Two-view multibody structure-and-motion with outliers through model selection. *PAMI*, 2006.

[79] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[80] SLAMcore. wheel odometry. `https://blog.slamcore.com/wheel-odometry-blog`.

[81] S. Song and M. Chandraker. Joint sfm and detection cues for monocular 3d localization in road scenes. In *CVPR*, 2015.

[82] S. Song and M. Chandraker. Joint sfm and detection cues for monocular 3d localization in road scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3734–3742, 2015.

[83] I. Spectrum. What is a robot?, May 2020.

[84] C. Stachniss, J. J. Leonard, and S. Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 1153–1176. Springer, 2016.

[85] G. P. Stein, O. Mano, and A. Shashua. Vision-based acc with a single camera: bounds on range and range rate accuracy. In *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No. 03TH8683)*, pages 120–125. IEEE, 2003.

[86] N. Sünderhauf. *Robust optimization for simultaneous localization and mapping*. PhD thesis, Technischen Universitat Chemnitz, 2012.

[87] P. Teunissen. Nonlinear least squares. 1990.

[88] I. Ullah, X. Su, X. Zhang, and D. Choi. Simultaneous localization and mapping based on kalman filter and extended kalman filter. *Wireless Communications and Mobile Computing*, 2020, 2020.

[89] L. J. Van Vliet, I. T. Young, and G. L. Beckers. A nonlinear laplace operator as edge detector in noisy images. *Computer vision, graphics, and image processing*, 45(2):167–195, 1989.

[90] R. Vidal, Y. Ma, S. Soatto, and S. Sastry. Two-view multibody structure from motion. *IJCV*, 2006.

[91] O. R. Vincent, O. Folorunso, et al. A descriptive algorithm for sobel image edge detection. In *Proceedings of informing science & IT education conference (InSITE)*, volume 40, pages 97–107, 2009.

[92] R. Wang, M. Schworer, and D. Cremers. Stereo dso: Large-scale direct sparse visual odometry with stereo cameras. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3903–3911, 2017.

[93] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *CVPR*, 2019.

[94] C. Wojek, S. Roth, K. Schindler, and B. Schiele. Monocular 3d scene modeling and inference: Understanding multi-object traffic scenes. In *European conference on computer vision*, pages 467–481. Springer, 2010.

[95] Q. Xu and D. Ma. Applications of lie groups and lie algebra to computer vision: A brief survey. In *2012 International Conference on Systems and Informatics (ICSAI2012)*, pages 2024–2029. IEEE, 2012.

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

[97] W. Zhou, E. Shiju, Z. Cao, and Y. Dong. Review of slam data association study. In *Proceedings of the 2016 International Conference on Sensor Network and Computer Engineering, Xi'an, China*, pages 8–10, 2016.