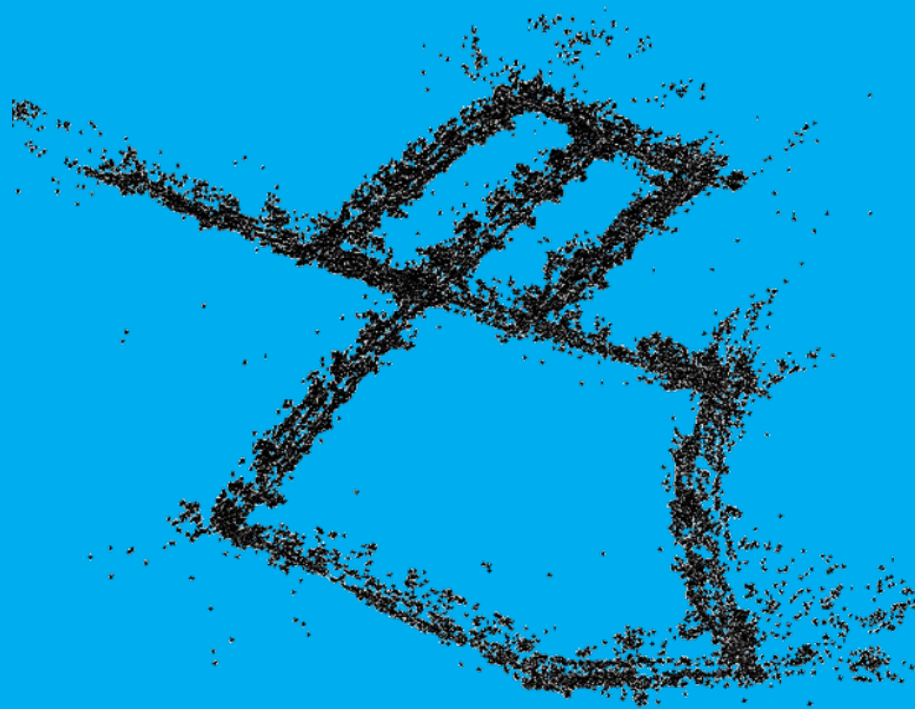


Robust map building for robot navigation in dynamic environments

Master thesis project

Prakash Radhakrishnan



Robust map building for robot navigation in dynamic environments

Master thesis project

by

Prakash Radhakrishnan

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday September 22, 2021 at 13:00 .

Student number: 5007224
Project duration: November 15, 2020 – September 22, 2021
Thesis committee: Prof. dr. ir. Wei Pan, TU Delft, Chair and supervisor
Prof. dr. ir. Javier Alonso Mora, TU Delft, External Committee
Ir. Yujie Tang, TU Delft, Daily supervisor

An electronic version of this thesis is available at
<http://repository.tudelft.nl/>.

Acknowledgments

This master thesis marks the completion of the number of credits required to obtain the Masters in Mechanical Engineering under the track 'Vehicle Engineering' at the Delft University of Technology.

I would like to thank my supervisor, Dr. Wei Pan, for allowing me to work on an exciting graduation project. I express my sincere gratitude to the daily supervisor Ir.Yujie Tang for her timely support and guidance throughout my work.

I would like to share warm greetings with my family and friends for their unconditional support during this journey.

Finally, I would like to acknowledge all the beautiful souls who motivated and helped me during this master program.

*Prakash Radhakrishnan
Delft, September 2021*

Abstract

Simultaneous Localisation and Mapping (SLAM) provide a novel solution for the robots to localise and navigate an unknown environment. Initial SLAM research focused mainly on the indoor environment, assuming the background to be primarily static. In contrast, the real world has dynamic interactions that restrict the implementation of SLAM to limited scenarios. This brings a higher requirement to deal with the moving objects in dynamic environments for robust SLAM performance.

Semantic understanding of the environment helps in filtering out the influence of dynamic objects in the vicinity. An instance segmentation based on two-stage neural architecture is used for this purpose, which is hard to operate in real-time navigation. In this project, the benefits of single stage neural architecture are studied in terms of speed and accuracy for improving the efficiency of dynamic features removal in the application of SLAM.

Although Instance segmentation architecture helps to identify the potentially dynamic object by learning from the dataset, it cannot differentiate moving objects from non-moving objects in the dynamic class. Hence, all the features corresponding to the predicted dynamic class are removed even when the objects remain stationary, affecting the quality of SLAM performance. A two-stream encoder-decoder architecture is developed to segment the moving masks using RGB and optical flow input, improving feature tracking without affecting robustness. The feasibility of encoding dynamic information to enhance quality semantic mapping is also studied.

Contents

Acknowledgments	iii
Abstract	v
List of Figures	xii
List of Tables	xiii
1 Introduction	1
1.1 Motivation	2
1.2 Research question	2
1.3 Contributions	3
1.4 Overview	3
2 Related work	5
2.1 SLAM architecture	5
2.1.1 Direct method	5
2.1.2 Feature-based method	6
2.2 Dynamic objects	8
2.2.1 Geometrical approaches	8
2.2.2 Learning based approaches	10
2.3 Review on deep learning architectures	11
2.3.1 Object detection	11
2.3.2 Segmentation	12
2.3.2.1 Semantic segmentation	12
2.3.2.2 Instance segmentation	13
3 Methodology	17
3.1 Methodology overview	17
3.1.1 Classification of dynamic objects	17
3.1.2 Stages of development	18
3.2 Tracking	19
3.2.1 Base - ORB-SLAM2	19
3.2.2 Stage 1 - Dynamic object segmentation	22
3.2.2.1 SOLOV2 network architecture	22
3.2.2.2 Combining dynamic object segmentation with ORB-SLAM2	24
3.2.3 Stage 2 - Moving object segmentation	25
3.2.3.1 Two stream network architecture	25
3.2.3.2 Combining moving object segmentation with ORB-SLAM2	26
3.3 Mapping	27

3.3.1	Dynamic semantic mapping	28
3.3.2	Instance semantic mapping	29
3.3.2.1	Multi-object tracking	29
3.3.2.2	Combining multi-object tracking with ORB-SLAM2	30
3.3.3	Dynamic density estimation	31
4	Dataset and metrics	33
4.1	Dataset	33
4.1.1	SLAM	33
4.1.2	Stage 1 - Segmentation dataset	34
4.1.3	Stage 2 - Moving object dataset	35
4.1.3.1	New dataset generation	36
4.2	Metrics	36
4.2.1	Absolute trajectory error - SLAM	36
4.2.2	IOU estimation - Segmentation	37
5	Experimentation and results	39
5.1	Object segmentation and tracking models	39
5.1.1	SOLOV2 - Dynamic object segmentation	39
5.1.1.1	Result and discussion	40
5.1.2	Two stream architecture - Moving object segmentation	41
5.1.2.1	Result and discussion	42
5.1.3	Multi-object tracking	45
5.1.3.1	Selection of feature embedding model	46
5.1.3.2	Result and discussion	47
5.2	Experiments on SLAM	48
5.2.1	Tracking	48
5.2.1.1	Base - ORB-SLAM2	48
5.2.1.2	Stage 1 - Dynamic object segmentation	49
5.2.1.3	Stage 2 - Moving object segmentation	50
5.2.2	Mapping	53
5.2.2.1	Dynamic Semantic mapping	54
5.2.2.2	Instance semantic mapping	55
5.2.2.3	Dynamic density estimation	55
5.2.2.4	Map management	56
5.3	Real-world validation	58
6	Conclusion	61
6.1	Recommendations	62
A	Appendix	63
A.1	SLAM Background	63
A.1.1	Feature matching	63
A.1.2	Coordinate transformations	64
A.1.3	Epipolar constraint	65
A.1.4	Triangulation	66
A.1.5	Bundle adjustment	66

A.1.6	Graph based SLAM	66
A.1.7	Optical flow	67
A.2	Deep learning Background	68
A.2.1	Neural networks	69
A.2.1.1	Neuron	69
A.2.1.2	Activation function	70
A.2.1.3	Loss function	71
A.2.1.4	Optimiser	71
A.2.2	Convolutional Neural Network	73
A.2.2.1	Convolution and its properties	73
A.2.3	Pooling	73
A.2.4	Stride and Padding	74
A.2.4.1	Upsampling	74
B	Appendix	75
B.1	Geometrical approach - Stage 2	75
B.2	Semi-dense Mapping	76
	Bibliography	88

List of Figures

2.1	Direct and feature based SLAM	6
2.2	Degenerative motion	9
2.3	Deep learning on feature extraction	10
2.4	Two stage detectors overview	12
2.5	Mask-RCNN	14
2.6	Comparison of instance segmentation architectures	14
3.1	Dynamic classification	17
3.2	Stage 1 classification	18
3.3	Stage 2 classification	19
3.4	Components of ORB-SLAM2	20
3.5	SOLOV2 mask head	23
3.6	Two stream architecture	26
3.7	Semantic mapping	28
3.8	Multi-object tracking	30
3.9	Track image representation	30
4.1	Ground truth of KITTI odometry dataset	34
4.2	KITTI segmentation dataset	35
4.3	KITTIMoSeg refinement	35
4.4	Moving object segmentation dataset generation	36
4.5	Trajectory alignment	36
4.6	IOU metrics	38
5.1	Quantitative comparison - SOLOV2 and Mask-RCNN	40
5.2	Qualitative comparison - SOLOV2 and Mask-RCNN	41
5.3	Loss function BCE loss vs dice loss	42
5.4	Moving object segmentation - Result	42
5.5	Moving object segmentation - Analysis	43
5.6	Flow image on pure translation	43
5.7	Heat map of moving object in the dataset	44
5.8	Flow image without camera motion	44
5.9	Flow image on pure rotation	44
5.10	Moving object segmentation - Refinement	45
5.11	Implementation of tracking module	46
5.12	Selection of feature embedder model	47
5.13	Multi-object tracking - Result	47
5.14	ROS implementation of our pipeline	48
5.15	Tracking error comparison - Baseline vs Stage 1	49
5.16	Tracking result - Stage 1	50
5.17	Limitation of Stage 1 - Kitti sequence 05	50

5.18 Tracking result - Stage 2	51
5.19 Individual trajectory comparison.....	52
5.20 ATE plot - Base vs Stage1 vs Stage2	52
5.21 Base static map	53
5.22 Dynamic semantic map - Stage 1	54
5.23 Dynamic semantic map - Stage 2	54
5.24 Instance semantic map	55
5.25 Low dynamic interaction example	56
5.26 Dynamic density estimation	56
5.27 Octomap with different resolution	57
5.28 Robot setup	58
5.29 Real world experiment result - Tracking and mapping	59
A.1 Epipolar geometry	65
A.2 Graph based SLAM	67
A.3 Optical flow explanation.....	68
A.4 Optical flow representation	68
A.5 Activation functions.....	70
A.6 Learning rate explanation	72
A.7 Max and average pooling	74
A.8 Qualitative comparison.....	74
B.1 Individual feature tracking using flow vectors	75
B.2 Epipolar line with tracked features	76
B.3 Semi-dense mapping - ORB-SLAM2 implementation	77
B.4 Semi-dense mapping results	77

List of Tables

4.1	KITTI odometry dataset	34
5.1	Parameter setting - Motion segmentation model	41
5.2	Absolute trajectory error - Stage 1 experiments	49
5.3	Absolute trajectory error - Stage 2 experiments	51
5.4	Map memory management	57

1

Introduction

The autonomous robot uses perception to understand the surrounding environment and performs actions based on planning and control. Robots' autonomy opens new possibilities in various applications, including unmanned ground vehicles, drones, and even underwater robots. The primary task of an autonomous robot is to localize itself in the given space. Currently, most outdoor applications localize using GPS signals [1] and pre-built maps available in the system. However, the GPS signal is not robust due to disturbances in closely spaced buildings, tunnels and dense plantations. Simultaneous Localisation and Mapping (SLAM) is a well-known solution for locating and navigating an unknown environment by making the system operate independently with its learnt representations of surroundings. SLAM helps in tracking the poses of the ego-vehicle and maps the surrounding environment space. The versatility of SLAM usage comes with its flexible nature to adapt to various scenarios.

Simultaneous Localization and Mapping perceives the data from the exteroceptive sensors like camera, lidar, radar and IMU. Exteroceptive sensors acquire measurements that contain meaningful representations of the surroundings. SLAM usually process data from one of the sensors or in some case it fuses information from multiple sensors to generate the final output. We focus on visual SLAM, which uses a camera as the primary sensor modality. Cameras are passive sensor that is relatively cheap and consumes less power for its operation. Monocular cameras are a minimalistic visual sensor configuration with much practical application but suffer heavily from scalability issues. It captures the three-dimensional space into a two-dimensional image by losing depth information. Nevertheless, visual features are always challenging to process compared to the other sensors. Our study focuses on simultaneous localization and mapping using monocular cameras.

The main challenges of SLAM are loop closure, initialization, semantic reasoning and handling dynamic objects in the environment [2]. Visual SLAM research has evolved to address the initialization and loop closure issues. On the other hand, dynamic objects are dealt with using the geometrical method that produces satisfactory outcomes. After the advent of deep learning in computer vision, understanding semantics from the visual features are learnt at ease. Artificial intelligence(AI) models achieve accurate and faster inference, which makes them popular across various domains. The

semantics of the environment learnt from the AI model is used to identify and isolate the dynamic objects to enhance the robustness in the SLAM functionality. This study tackles important limitations such as semantic reasoning and dynamic object handling by fusing the output of convolutional neural networks in SLAM to enhance tracking and mapping results.

1.1. Motivation

Initial SLAM research mainly focuses on the indoor environment where the objects in the surroundings are static [3]. This static assumption might not be valid in the real world, causing inconsistent tracking and mapping results. High dynamic interaction in the outdoor environment aggravates the problem affecting the quality of tracking and mapping further. Thus, static assumption restricts SLAM to be implemented in limited scenarios. Semantic perception of the environment gives fundamental knowledge of the vicinity to deal with dynamic interactions [4]. The output of deep learning models (learning-based approaches) offers valuable insight on learning semantics for improving tracking and navigation by removing dynamic features. In addition to removing the dynamic object, it is vital to semantically associate the learned representation in 3D space to understand the map better. Hence, this study uses single-stage instance segmentation architecture combined with moving object detection and multi-object tracking to remove the dynamic objects in the tracking process and build a semantically meaningful instance map.

1.2. Research question

The objective of the study is to build a semantically meaningful map using SLAM techniques while managing dynamic interactions, which helps enhance robot navigation quality.

Our research question is,

‘ How to build robust interactive map handling dynamic objects that enhances the navigation quality of robots? ’

The following sub questions are framed from main research question,

- What is a popular approach to overcome challenges of dynamic objects in SLAM?
- How to improve the quality of tracking and mapping with dynamic interaction?
- How to build a semantically meaningful map?

1.3. Contributions

We answer the research questions by following contributions.

- **Single-stage instance segmentation architecture** - According to the literature study, the most advanced deep learning architecture used in SLAM is the two-stage instance segmentation architecture. In this project, the benefits of single stage instance segmentation architecture are studied and implemented for dynamic object identification in SLAM.
- **Moving object segmentation** - To distinguish moving objects from non-moving objects, individual object states in three-dimensional space are tracked in tightly coupled methods [5] which are computationally intensive. On the other hand, loosely coupled methods [6] based on a learning-based approach attain real-time performance on finding a particular class but fail to distinguish the moving object. In this work, the implementation of moving object segmentation unleashes the possibility of using loosely coupled methods directly to perform a similar task that helps attain real-time performance.
- **Semantic Mapping** - Semantic mapping addresses 2D-3D data association problems to improve the semantic understanding of the generated map. We create dynamic and instance-specific semantic mappings. The dynamic semantic map categorizes points as static, moving, or non-moving dynamic points. Instance semantic mapping detects and distinguishes individual instances of vehicles present in the environment.
- **Multi-object tracking** - Instance segmentation output does not correlate segmented objects between two frames, which prevents it from tracking the instances in the sequence. We implement a separate tracking module to track objects by associating the same objects in successive frames.

1.4. Overview

This outline of the report is as follows; [Chapter 2](#) summarizes the literature study and the critical choices arrived from it. [Chapter 3](#) walks through the learning-based methodology used in our research to improve tracking and mapping results of SLAM. [Chapter 4](#) explains the selection of the dataset and chosen metrics used to evaluate our procedures. [Chapter 5](#) concentrates on the experiments, results and discussion to justify the selection of architecture and showcases the improvement in SLAM with our implementation. [Chapter 6](#) winds up our study and suggest future direction on improving key modules. At last, the appendix reveals trials on the geometrical approach for moving object detection and semi-dense mapping, which are not included in our final pipeline. It also delves into essential concepts on SLAM and deep learning needed for comprehending the methods used in this study.

2

Related work

The objective of the chapter is to find the suitable SLAM architecture and discuss various approaches to overcome the challenges of dynamic interaction in the SLAM. The last part covers the widely used learning-based method and proves the empirical choice of single-stage instance segmentation for completing our pipeline. Detour to the [appendix A](#) would help brush up the basics before diving into the details of SLAM and deep learning models.

2.1. SLAM architecture

Visual SLAM architecture uses optical sensors such as Monocular, stereo and RGB-D cameras to acquire information about the surroundings. In this study, monocular cameras were kept as the prime focus. Monocular SLAM is categorised further into feature-based, direct and learning-based methods based on how the image is processed to attain the desired outcome on tracking and mapping.

2.1.1. Direct method

The direct method correlates the pixel intensity values across the frames to find correspondences and build a map. Direct methods can generate dense, sparse, and semi-dense maps based on the utilisation of pixel correspondence. An important problem of the direct method is the computational cost [7] and sensitiveness to noise due to illumination changes. Lens attenuation and gamma correction improve the robustness against noise. On the other hand, direct methods are very robust towards motion blur and camera defocus.

Dense Tracking and Mapping (DTAM) [8] is one of the direct methods which uses input from the handheld camera and process them using GPU to achieve real-time performance. The algorithm estimates inverse depth based on the photometric error to generate a dense map. Photometric error is the difference in image intensity between the same point in two image frames. The computation over entire pixels makes the algorithm run slower in the CPU setting. Hence, sparse and semi-direct methods naturally become salient for many applications to operate in higher frequencies. **Semi-Direct Visual Odometry (SVO)** [9] fuses the direct and feature-based method

to estimate visual odometry. Algorithm process image patches instead of entire pixels to reduce the computational cost. Photometric error is minimised to estimate the pose, and re-projection error is minimised to align the feature patches. Though mapping is similar to DTAM, depth is calculated in a probabilistic Bayesian fashion to reduce the complexity. The final map generated by SVO is sparser than DTAM. **Direct Sparse Odometry (DSO)** [10] is another direct method that produces a sparser map.

Large-Scale Direct Monocular SLAM(LSD-SLAM) [11] is a semi-dense direct method designed to achieve real-time performance in the CPU by processing images as keyframes in a graph-based outline. A keyframe is a representative frame that stores relative information of all the frames in a given time interval. A key ingredient of LSD-SLAM is to use the image gradient to recover edges which makes them robust to changes in intensity when compared with other direct methods.

Direct methods such as DTAM, DSO and SVO are relatively slow and do not support loop closure functionality. On the other hand, LSD-SLAM efficiently handles the global maps, which makes loop closure feasible. Hence, LSD-SLAM with semi-dense maps is a natural choice among other direct methods due to its ability to operate real-time and loop-closure.

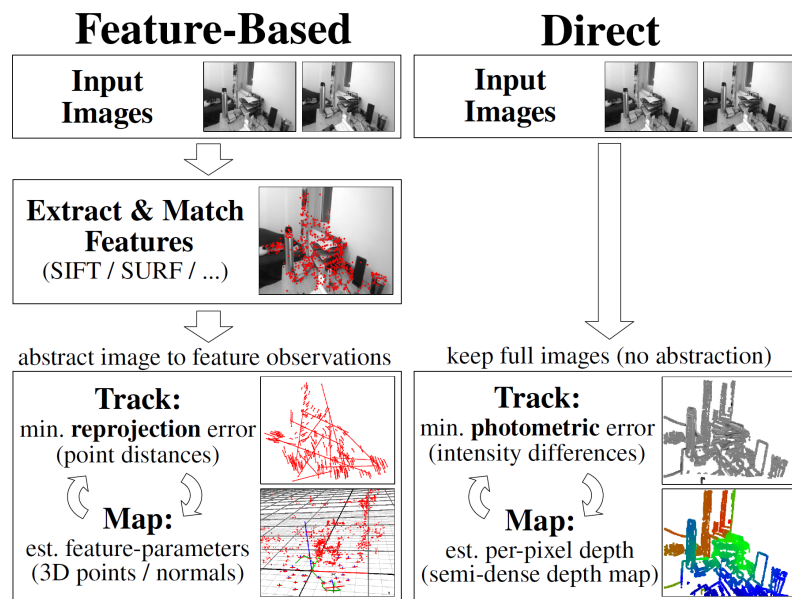


Figure 2.1: Comparison between direct and feature based method [12]

2.1.2. Feature-based method

Feature-based SLAM uses extracted visual features from the images for estimating pose and map 3D landmarks. Unlike the direct method that utilises pixel intensities, feature-based SLAM takes advantage of the geometry of the environment, making them robust to noises. Most of the feature-based process runs in real-time but creates a sparser map representation compared to the direct method.

Mono slam [13] is a feature-based filtering approach for monocular cameras that focuses mainly on localisation compared to mapping. Probabilistic state estimation is used to track the moving cameras and map landmarks. The map is initialised with known objects and updated sequentially after tracking. The filtering based approach encapsulates single probability distribution for all the estimated poses. The size of the covariance matrix explodes with an increase in features resulting in a quadratic rise of computational cost, which makes the algorithm restricts to a small-scale environment. **Parallel tracking and mapping (PTAM)** [14] is developed to achieve real-time performance by adding a parallel thread for tracking and mapping. Unlike sequential tracking and mapping adapted in MonoSLAM, parallelizability allows the maps to be optimised using bundle adjustment instead of incremental mapping. It is essential to observe that bundle adjustment still increases the computational cost based on the number of frames. Thus, usage of keyframes provides a trade-off between accuracy and computational cost. The important drawback of PTAM is that five-point stereo map initialisation is done based on the user input manually. In the advent of tracking failure due to changes in the environment, this might lead to map failure. PTAM explicitly developed to a small AR environment restricting them from large scale mapping. **ORB-SLAM** [15] is a successor of PTAM designed specifically for monocular cameras. ORB-SLAM contains three parallel threads for tracking, mapping, loop closure, and two modules explicitly for place recognition and map. **ORB-SLAM2** [16] is an improvised version of ORB-SLAM that could process stereo and RGB-D inputs in addition to monocular camera input. Unlike PTAM, ORB-SLAM2 allows automatic initialisation using two geometric models based on homography for a planar scene and a fundamental matrix for a non-planar scene. Heuristic calculation based on the equation 2.1 identifies whether the scene is planar. If $R_H > 0.45$, then homography is chosen for initialisation or vice versa. A detailed explanation of all the modules in ORB-SLAM2 is discussed later in the report.

$$R_H = \frac{S_H}{S_H + S_F} \quad (2.1)$$

The vital difference between ORB-SLAM2 with other feature-based methods is their ability to perform loop closure. ORB-SLAM2 is the best architecture among other feature-based methods due to its ability to run in real-time, reinitialise automatically and provide large scale mapping.

Conclusion - Choice of SLAM The learning-based approach supports visual odometry only. They do not provide a complete solution to SLAM and hence are not discussed in detail the report. Based on the survey of direct and feature-based method, the selection of SLAM architecture is narrowed down to LSD-SLAM and ORB-SLAM2 on account of their ability to run in real-time with loop closure functionality. In terms of versatility, ORB-SLAM2 extend its support to various visual sensors such as monocular camera, stereo camera, and RGB-D input, but LSD-SLAM baseline uses only monocular cameras. As our work is more focused on dynamic interaction, the SLAM algorithm must be suitable for both indoor and outdoor environments. Outdoor environments tend to have global illumination changes, which violates the assumption of brightness constancy used in LSD-SLAM. Also, based on the experimental comparison between ORB-SLAM2 and LSD-SLAM by [17], it is shown that the ORB-SLAM2

algorithm provides robust tracking results on popular odometry datasets. Hence ORB-SLAM2 algorithm is chosen as SLAM architecture in our pipeline.

2.2. Dynamic objects

After selecting the SLAM architecture, we aim to find an effective method to improve the robustness of SLAM results while managing the dynamic interaction of the environment. The issue of dynamic interaction is addressed by either eliminating dynamic moving objects as outliers or track the individual dynamic object state in its 3D representation. To narrow down the review based on the final choice of SLAM, this part will solely concentrate on studies related to ORB-SLAM2(feature based methods).

2.2.1. Geometrical approaches

In the conventional SLAM approach, the area of static entities are considered as background and dynamic objects as foreground. The moving object detection (**MOD**) method [18] match the extracted features of moving objects with the previous set of descriptors database created using prior knowledge. Three representative features of moving objects are selected, and the assumption of rigidity aids in determining their motion. It is a naive method since it can only identify objects based on a previous list of features; however, the representation of feature descriptions varies substantially in the actual world.

Some of the other studies deal with the background subtraction method [19] where basic polynomial constraints is used to detach moving objects from the stationary states. Researchers also use sparse labelling and rank restrictions to track points in the frame and computes a projection matrix to solve for inliers and outliers. The primary drawback of this point-based method is that it does not consider structure consistency in the calculation. The majority of background subtraction algorithms are only successful when the visual sensor is static and assumes the surrounding environment as a planar surface.

To deal with dynamic objects captured by freely moving cameras, researchers exploited the **epipolar constraint**. According to the constraint, static points always lie closer to the epipolar line whereas points located outside the epipolar line are classified as dynamic objects. A similar approach was used in [20], where, in addition to the epipolar restriction, a **Flow vector bound** was added to account for degenerative motion. Degenerate motion [21] occurs when the objects in the scene move parallel along the direction of the camera motion. Since, the points corresponding to the objects move along the epipolar line, making it harder for epipolar constrains to separate dynamic points. The flow vector bound defines how points translate in relation to the depth. By specifying upper and lower limits for depth values, the displacement of the points associated with those constraints may be calculated. A feature vector that extends beyond these displacement limits is termed dynamic. Inaccurate depth estimation precludes the use of flow vector bounds in monocular cameras.

Optical flow from the image sequence is compared with the predicted artificial op-

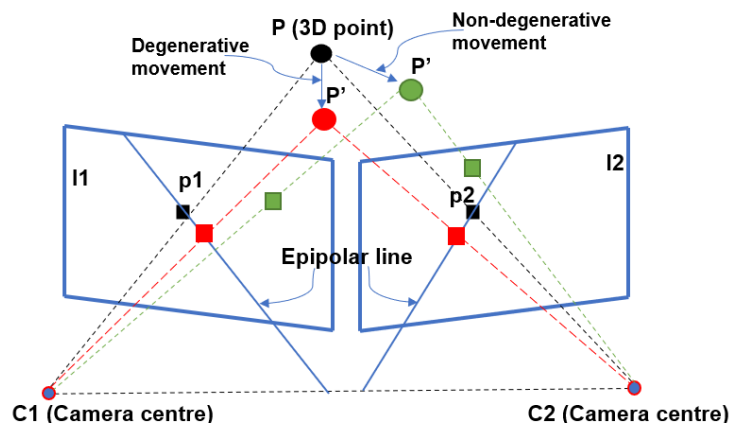


Figure 2.2: Degenerative vs non-degenerative motion

tical flow to determine the dynamic pixels in the study [22]. Artificial optical flow is estimated based on the relative pose variation acquired through homography during tracking. The generated optical flow is calculated based on the ego-motion of the camera, which influences the quality of prediction when the camera is moving. Disentangling the camera motion from rest of the object is essential for detecting moving objects in a scene, which is often accomplished by ego-motion compensation. **Ego-motion compensation using frame differencing** is experimented by [23] in which a list of feature correspondence is tracked to estimate the ego-motion. Frame differencing of an ego-compensated image with the current frame helps to identify potentially moving points. A similar idea of using compensated image in frame differencing along with particle filtering is subsequently experimented on RGB-D SLAM [24]. The main limitation of the frame differencing approach is the assumption of small parallax between image sequences. The algorithm does not work well in handheld cameras where the parallax could vary significantly.

Correlation between the generated map points is also used in [25] to eliminate the influence of dynamic objects in the surroundings. The relative position of any two points on the map is always consistent in temporal space; hence these points could be correlated. Such a correlation always exists between static points in 3D space. The technique uses graph initialization, edge culling, and static point determination to identify correlation represented as a sparse graph. Point correlation optimized using bundle adjustment and squared Mahalanobis distance to remove inconsistent edges. The graph with the greatest volume is considered static. However, when the ratio of moving objects in the scene exceeds the ratio of static objects, the system fails.

According to the above discussed methods, it is apparent that geometrical approaches make several assumptions to locate dynamic points. These assumptions are often violated in real-world situations. For example, the optical flow assumption of constant brightness and velocity smoothness across adjacent pixels does not hold in the real world. In the case of ego-motion compensation and point correspondence, the background is required to be larger than the foreground, which might not be possible in the actual world.

2.2.2. Learning based approaches

To overcome the shortcomings of the geometrical method, the learning-based technique is used to eliminate dynamic objects from the image frame using neural architecture. The main idea is to integrate the output of deep learning models with SLAM, thereby excluding features that fall inside the predicted area of the model output, considering them to be dynamic. A learning-based approach could be applied either using object detection or a segmentation architecture to remove the dynamic associations. Small detour to next section would help in understanding the details of various convolutional neural architectures clearly. This section reviews various studies conducted on SLAM using learning based approaches.

Object detection model YOLO [26] adapted by [27] for predicting dynamic classes, and features extracted inside the predicted bounding box are removed considering them as moving. However, some of the static zones which are not part of dynamic objects are also included inside the bounding box as shown in the figure 2.3. Exact contours identification of the object boundary is needed to isolate the dynamic areas. These contours were usually estimated using depth histograms which could not be directly applied in monocular cameras due to the inconsistent depth estimation. PO-SLAM [28] applies a similar technique to segment contours in YOLO prediction using depth histograms. The major disadvantage of utilizing object detection is that it necessitates using a separate algorithm to segment the object's mask, which steers the research towards semantic segmentation architecture.

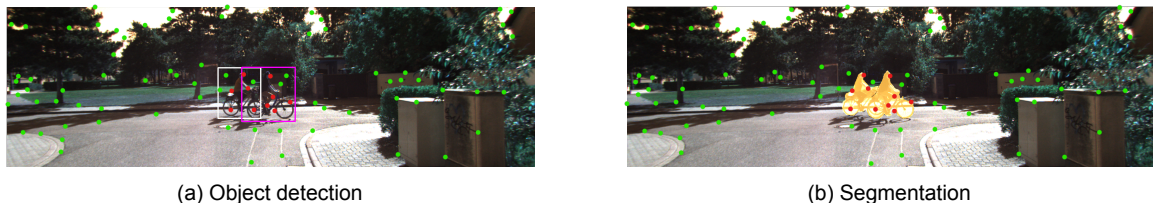


Figure 2.3: Deep learning on feature extraction

The **Semantic Segmentation** is a simple and ready to use technique in SLAM since it avoids contour estimation. For the same model complexity, the inference time of semantic segmentation is slightly longer than that of the object detector. Taking this into consideration, the shallow semantic network such as ENet [29] is usually adapted in ORB-SLAM2 to segment the dynamic classes in real-time [30]. Recent research has merged learning-based and geometric approaches to remove dynamic classes that neural networks have not learned efficiently. **DS-SLAM** [31] combines SegNet architecture [32] along with moving consistency (based on epi-polar constraint) to segment *pedestrians* from an image. **SOF-SLAM** [33] extracts feature correspondence of moving objects using semantic segmentation and optical flow. While the basic idea of SOF-SLAM is similar to that of DS-SLAM, the former handles many classes.

While semantic segmentation is superior to object detectors, it lacks the ability to recognize individual instances. To enhance pixel-level categorization and instance identification, Mask-RCNN [34] is used by SLAM researchers. In the study [35], Mask-RCNN is amended to segment the output of a dynamic mask. The output of

Mask-RCNN is not quite accurate along the boundaries of the predicted mask. To improve the accuracy along the edges, contour refinement using canny edge detection is used. Canny edge detection refines edges by calculating moments between the contour centroid and the edge centroid. Additionally, the **Dyna-slam** [36] study utilises Mask-RCNN as a priori for detecting dynamic objects. The study also uses multiview geometry utilising parallax angle (projection of key points from the previous frame to the current frame) and depth to segment the moving objects in the current scene as a region growth algorithm.

Conclusion Segmentation is preferred to the object detection as a separate module is needed to extract the shapes of the objects inside the bounding boxes. Under segmentation, semantic segmentation architectures such as segNet and ENet used in SLAM achieve a faster inference but are not sufficiently precise. On the other hand, instance segmentation using Mask-RCNN is not optimized for real-time applications. Though the learning-based method delivers promising results, it is necessary to perform a careful study to select an architecture that provides a favourable tradeoff between speed and accuracy. The following section discusses various convolutional neural architectures used in the computer vision domain.

2.3. Review on deep learning architectures

Deep learning techniques such as recurrent networks, convolutional neural networks (CNNs), and a few generative networks aid in the solution of several computer vision tasks such as detection, localization, and segmentation. This chapter highlights the various computer vision tasks for predicting moving objects using a convolutional neural network and aims to select the state of the art model for our experiments.

2.3.1. Object detection

Object detection is a technique that uses bounding boxes to locate an object in a given picture. A substantial evolution of deep learning started after the onset of AlexNet [37] which classify 1000 different classes in ImageNet challenge. Two significant categories of object detectors are two-stage and single-stage detector. Only the architectures which are used in SLAM research were reviewed in detail.

The **Two stage object detector** detects objects in two stages: the first stage generates region proposals, and the second step predicts the class for the generated proposals. Region proposal by selective search algorithm is adapted in **R-CNN** (Regions with CNN features) [38]. Since the proposals were chosen using a preset method, it is possible to have inadequate recommendations. **Fast-RCNN** [39] process each candidate feature proposal separately, bounding box and class predictions are obtained jointly from the proposals. However, the model had poor inference due to the external region proposal approach, preventing Fast-RCNN from being used in real-time. **Faster-RCNN** [40] overcomes the issue of external region proposals by implementing Region Proposal Network (RPN), which uses a convolutional neural network to predict possible proposals. The proposals are reshaped to a fixed size using ROI pooling, and the objectness score for each proposal is estimated using RPN to pre-

dict five outputs (4 for rectangular boxes and 1 for score). Non-maximal suppression and cross-boundary implementation were adapted to remove the redundant results. Faster-RCNN runs much faster than previous methods; however, it suffers from redundant computation with expensive training procedures at each stage. Mask-RCNN used in one of our experiment inherits properties from Faster-RCNN. While Faster-

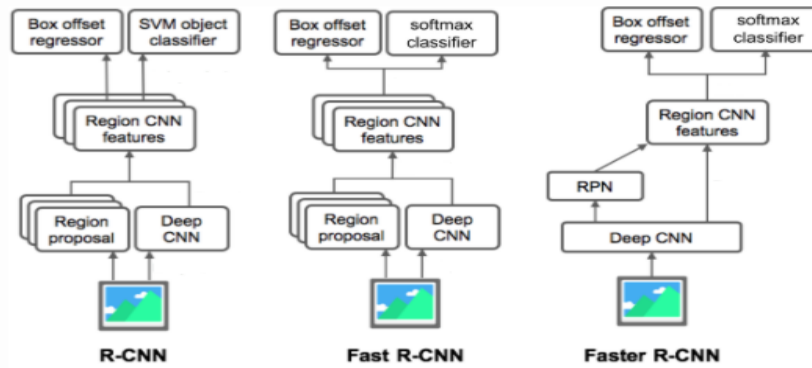


Figure 2.4: Overview of two stage object detectors [41]

RCNN has a high degree of accuracy, it is computationally intensive. The introduction of single-stage detectors resolves this computation problem by automatically predicting bounding boxes and class categories from the picture without a separate region proposal network. The design of a single-stage detector allows end-to-end optimization during the training. You Only Look Once (YOLO) is a single-stage object detector that predicts the output based on a global understanding of the picture. This global representation eliminates false positives (a common issue with two-stage detection) and generalizes the model well on unseen data. YOLO divides the input image into grids, and each grid is then responsible for detecting the presence of an object. The inclusion of a spatial restriction in terms of grid boxes prevents YOLO from anticipating smaller items within the trained class. To address this problem, the output is predicted at various scales in designs such as Single Shot MultiBox Detector [42], and YOLOv3 [43].

2.3.2. Segmentation

The segmentation network predicts an output label per pixel in the given images. The segmentation network is categorized into semantic segmentation and instance segmentation based on its ability to estimate object instances.

2.3.2.1. Semantic segmentation

Semantic segmentation categorizes the object boundaries and predicts class labels for each pixel in an image. The final prediction of the network is interpreted as a cluster of semantically meaningful labels. Supervised semantic segmentation architectures are grouped as feature-enhancement-based, deconvolution-based, and context-based approaches depending on the information flow for predicting the final semantic output [44].

Feature enhancement techniques restore spatial characteristics in the network by allowing feature information to flow from previously extracted feature levels to deeper layers. Skip connections allow the semantics of shallow layers to be combined with the semantics of deeper levels, helps in preserving object location and edges. It also helps to restore the loss of spatial information caused by max-pooling layers. Fully Convolutional Networks (FCN) [45] and UNet [46] are two popular architectures that employ feature enhancement approaches. Concatenating features from previous layers raises the memory requirement, which is one of the significant drawbacks of this method. **Deconvolution-based** semantic segmentation adapts encoder-decoder architecture where pooling indices at each encoding stage are saved and retrieved in the decoder stage to overcome the memory requirement issue faced by feature enhancement methods. The SegNet architecture [32] implements the deconvolution technique for semantic segmentation across the encoder and decoder architecture. The use of pooling indices in the max-pooling step considerably decreases the trainable parameters and increases the inference speed, making it suited for various SLAM tasks. **Context-based** approach enlarge the receptive field of neural networks to perceive characteristics on global scales. Global average pooling and dilated convolution are used commonly to enlarge the receptive field of neural networks. Pyramid Scene Parsing Network (**PSPNet**) [47] retrieves the global context by aggregating the features from pyramid pooling modules (stack of features in various scales) to produce semantic segmentation. Global scaling improves segmentation in complicated situations, creates a connection between different classes, and reduces scaling error.

Semantic segmentation does not offer a clear difference between the object's instances, which is required to enhance map robustness. Instance segmentation architecture is adapted to address the need of comprehending object instances.

2.3.2.2. Instance segmentation

The important distinction between instance segmentation architecture and semantic segmentation architecture is that instance segmentation architecture identifies individual object locations for each pixel. In instance segmentation, there are two primary methods used, namely top-down and bottom-up approaches. The top-down method identifies objects using bounding boxes and then generates a segmentation mask. On the other hand, bottom-up approaches establish relationships between pixels from the same instance that are processed as embedding vectors.

Mask-RCNN [34] is a popular instance segmentation architecture employed in SLAM applications that adopts a top-down method. It is built on Faster-RCNN (two-stage object detector), adding a parallel branch for mask prediction. The masking branch process each feature output using a fully convolutional layer to produce a binary mask. The primary issue with Mask-RCNN is misclassification and incorrect masking as a result of the fixed mask resolution. In addition, the computational cost owing to the extra masking branch is quite high.

On the other hand, single-stage instance segmentation models improve the performance of instance segmentation tasks by achieving real-time inference. Only the

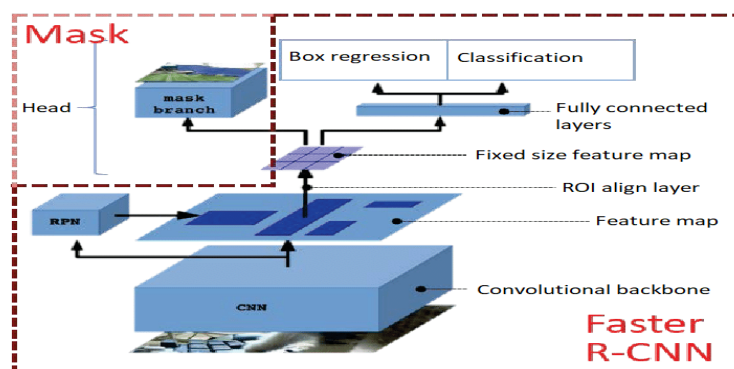


Figure 2.5: Mask-RCNN architecture

architecture that achieves real-time inference speed is reviewed here. Based on figure 2.6, YOLACT [48] generates output mask using two parallel tasks by generating the set of prototype masks and predicting per instance mask coefficients. Though YOLACT is fast, it suffers from misalignment of bounding boxes and localization failure in the presence of many objects resulting in reduced accuracy.

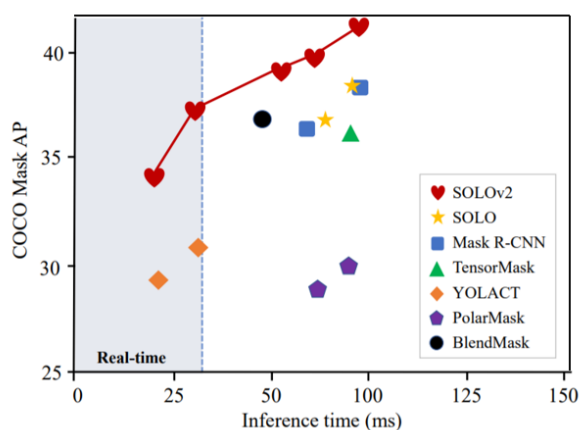


Figure 2.6: Speed vs accuracy comparison - instance segmentation architectures [49]

SOLO [50] is another significant single-stage instance segmentation design that predicts the instances from an entirely new perspective. It splits an image into grids and associates each cell for generating class, instance and mask. SOLOV2 [49] is a successor of SOLO, which use dynamic convolution and matrix non-maximal suppression to achieve faster inference. Based on the comparison of popular instance segmentation architecture as shown in figure 2.6, it is clear that SOLOV2 could achieve faster inference without compromising the segmentation accuracy.

Conclusion - While various deep learning architectures have been suggested, segmentation architecture has successfully integrated dynamic object information with SLAM. As our research question concentrates on generating an interactive map, we adapt an instance segmentation model to improve the tracking and mapping. A theoretical comparison among the common instance segmentation architectures reveals

that SOLOV2 is suitable to our study for dynamic object removal. Network architecture and implementation details of SOLOV2 are explained in detail in the following chapters.

3

Methodology

The chapter explains the overall implementation of the various methodology adapted in the project to improve the tracking and mapping results of ORB-SLAM2 by efficiently handling the dynamic objects in the scene. The structure of the chapter first introduces a basic outline of our implementation in the methodological overview 3.1 and then explains systematical inclusion of the proposed methods with ORB-SLAM2 to study the behaviour in tracking and mapping results.

3.1. Methodology overview

3.1.1. Classification of dynamic objects

The core of this project is to build robust map for robot navigation in dynamic environment. Firstly, our research is devoted to managing dynamic objects in the scene and the improvement is studied by observing the tracking result. Elimination of dynamic objects negates the tracking failure which inherently improves the mapping performance. So, it is essential to understand the basic categorization of dynamic classes used in our studies using an example.

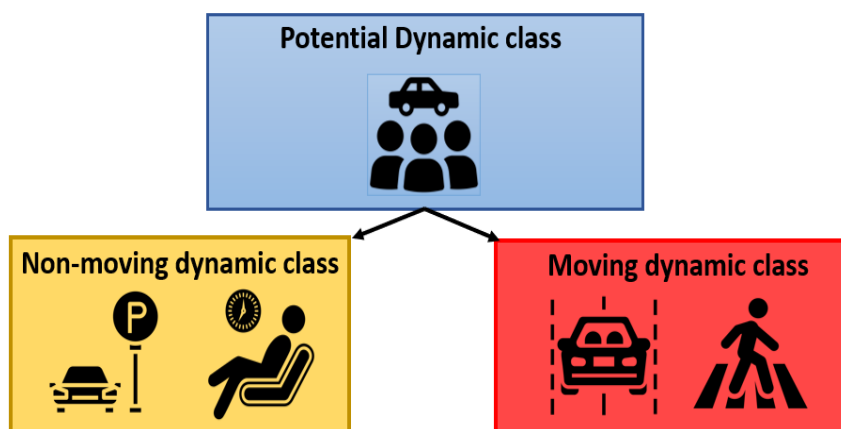


Figure 3.1: Classification of dynamic objects

Let us consider that a deep learning model is used to predict the dynamic classes from an RGB image. For instance, vehicles and pedestrians are two dynamic classes that would possibly move within the environment. In the project, we classify the classes that have ability to move as potentially dynamic objects based on our previous knowledge. However, the generic CNN model could not determine the moving vehicle separately. To realize predefined classes movement in the scene, we should divide potentially dynamic objects into moving and non-moving dynamic classes. In a nutshell, all the cars observed in the scene is considered as potentially dynamic class. While parked cars are sub-categorized as non-moving dynamic objects and the cars that are actually moving in the current instance are sub-classified as moving dynamic objects.

3.1.2. Stages of development

The primary objective of our approach is to answer the research questions 1.2 with a suitable methodology. To build interactive map, we need to improve both tracking and mapping results. Initially the tracking quality is studied in dynamic interactions where the learning based method reviewed in the chapter 2 is implemented with baseline ORB-SLAM2. The deployment of these learning based methods are done in three distinct stages. Each step is developed in response to the limitations of the preceding stage.

The summary of each development step that contributes to the handling of dynamic objects is as follows:

- **Base** - Base implementation of ORB-SLAM2 architecture without any modifications.
- **Stage 1** - This stage uses a single-stage instance segmentation architecture in association with ORB-SLAM2 to understand the semantics of all potentially dynamic classes. In practice, SLAM would process the features into two groups of points: static (e.g. buildings) and potential dynamic objects (e.g. all the cars in the scene).

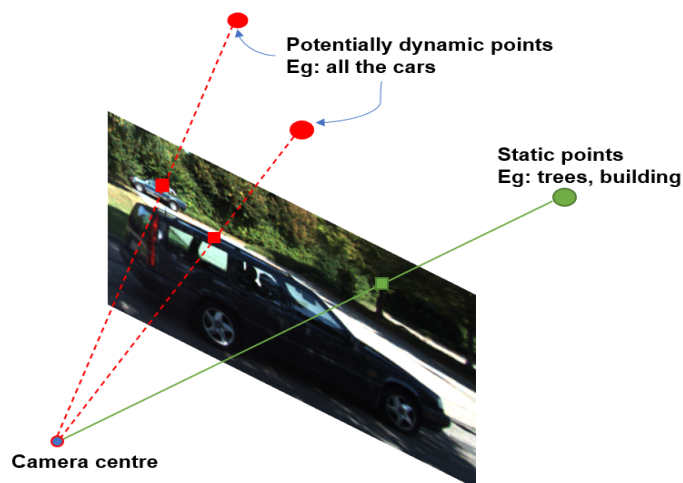


Figure 3.2: Stage 1 - Dynamic classification

- **Stage 2** - This stage incorporates moving object segmentation(not reviewed in chapter 2) over stage 1 implementation to distinguish moving and stationary objects. Overall, in the final process have three subsets namely static (e.g. Buildings), moving dynamic class(e.g. moving cars) and non-moving dynamic class (e.g. parked cars) as shown in figure 3.3

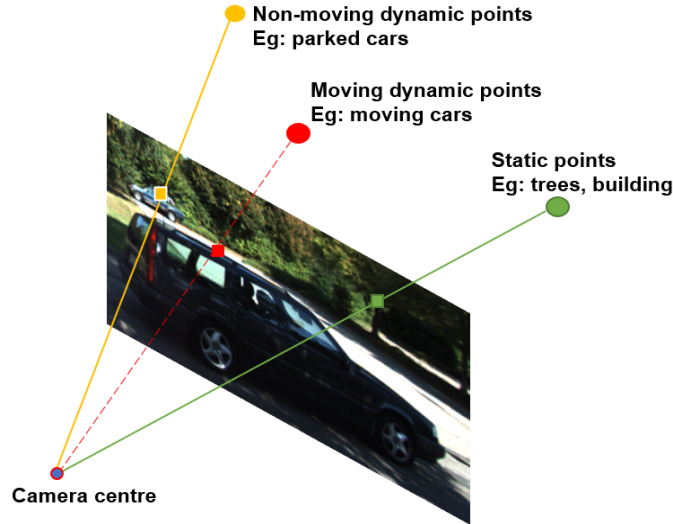


Figure 3.3: Stage 2 - Dynamic classification

Finally, our approach is directed towards strengthening the quality of maps by semantic reasoning. Our study experiments a concept of semantic mapping by efficiently using the learnt features from stage 1 and stage 2 implementation by generating dynamic and instance semantic maps. Output from the models that predicts dynamic classes could be directly used for generating dynamic semantic maps. However, the implementation of instance semantic segmentation involves the utilisation of the multi-object tracking in the segmentation output. Additionally, we compute a dynamic density to determine the amount of traffic in a given region based on the dynamic map. Lastly, map management is presented to store the global maps as point clouds and octomaps.

3.2. Tracking

3.2.1. Base - ORB-SLAM2

ORB-SLAM2 algorithm comprises of tracking, mapping and loop detection module. These module runs in parallel on three different thread to estimate visual odometry and map the environment. ORB-SLAM2 is a graph-based system that optimizes pose graph of keyframes by reducing re-projection error to generate landmarks. The functioning of graph based system is explained in the appendix. In addition to three parallel threads, each thread operates based on the subclasses such as frame, frame drawer, keyframe, keyframe database, initializer, map point, map drawer, ORB extractor, ORB matcher, solvers, and viewer. All the classes are inter-linked, i.e., any change to one has a detrimental effect on the output. Hence, modifications on the class function

during our implementation is done carefully to ensure that base functionalities of the SLAM are not jeopardized. We are primarily concerned with modules that contribute to final tracking and mapping in this project; no changes are made to the loop closure thread.

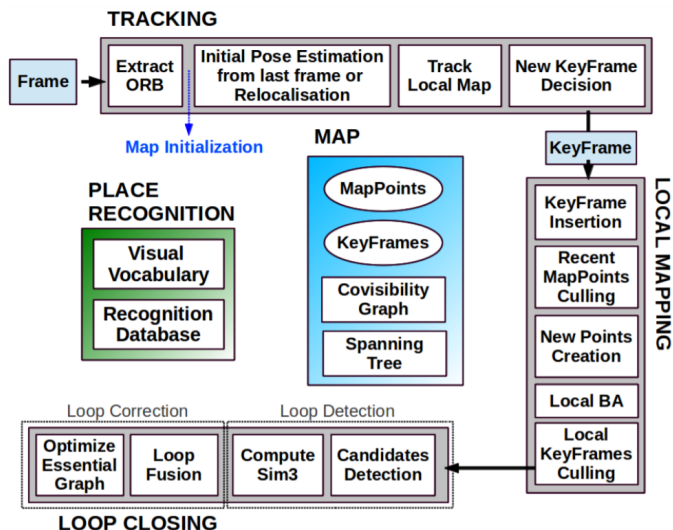


Figure 3.4: Components of ORB-SLAM2 [16]

Tracking thread: The tracking thread in ORB-SLAM2 is responsible for the estimation of visual odometry. Primarily it extracts features from individual frames and uses a perspective-n-point algorithm to estimate the camera poses. The tracking thread comprises of motion model, reference keyframes and relocalization module to track the camera instances. The motion model assumes the constant velocity constraint to predict the feature points based on the previous frame. The tracking is marked as successful or unsuccessful based on the number of inliers. The motion model works quite well only if the relative change between poses is minimal. When the motion model fails to track, reference keyframes are used in tracking. Visual features based on a bag of words are generated for the current frame compared to the reference keyframe features to estimate poses. Local bundle adjustment is used to rectify these postures. In certain situations, tracking might completely fail. To relocalize the current camera position in the advent of tracking failure, feature matching is performed across all the keyframe. One with the highest similarity score was chosen to estimate the current pose based on the perspective-n-point algorithm.

In addition to the estimation of camera poses, the tracking module search for new matches based on previous landmarks of the local map. These local maps are created using a collection of keyframes that shares the similar features of the current frames. The 3D points from the local maps are projected onto the current image frame and added as new matches if the viewing angle and distance from the camera centre are under a specified threshold. Recent matches are then optimized to refine the estimated poses.

The tracking module also superintends the selection of new keyframes based on the following condition.

- Gap between the last relocalization and current frame is at least 20
- Gap between the last keyframe and current frame is at least 20
- Current frame tracks at least 50 key points
- Local map thread is idle
- Similarity of points between the current frame and reference keyframe should be less than 90%

Mapping thread: The mapping thread is responsible for maintaining keyframes and map points. After a tracking thread selects a new keyframe, the mapping thread inserts it into the covisibility graph and ensures that visual features corresponding to the keyframes are updated appropriately. The covisibility graph is a layout where the nodes correspond to keyframes, and the edges represent map points. Map points and keyframes were added generously by the tracking thread. The map culling module monitors newly added points and retains them in the final map only if the created points could be observed in three keyframes, and the tracking module detects the points in 25% of frames. As map point culling, keyframe culling eliminates redundant keyframes when 90% of its associated map points is observed in at least three more keyframes. The culling process in the mapping thread assists in limiting the growth of the graph.

The main functionality of mapping thread is the ability to insert new map points, which results in a denser final map. Unmatched features in the current frame are examined in the other keyframe for potential matches to triangulate as a new map point. New map points should have depth, re-projection error, and scale within the specified limits in addition to satisfying the epipolar requirement. Local bundle adjustment optimizes the pose of the current keyframe and its neighbouring keyframes along with new map points associated with those keyframes.

Loop closing thread: The loop closure module recognizes the features of the regions seen by the system in the previous instance that helps the system correct the drift during tracking and mapping. The candidate frames for loop closure are chosen based on the predicted similarity score in the covisibility graph. The similarity score is determined by comparing the keyframe's bag of words representations. Similarity transformation with maximum inliers picked as a representative loop closure frame. The transformation matrix with seven degrees of freedom generated during the loop closure optimization corrects the current keyframe. It propagates the adjustment to all adjacent keyframes until the whole graph is corrected.

Theoretical representation:

Let Z , X , and O denote the camera measurement, the camera state, and the object state, respectively. l denotes the estimated 3D landmark generated by SLAM. Let w , c , and q symbolize the coordinates expressed with respect to world, camera, and

object at any point in time i . Additionally, the superscript and subscript o imply the predicted class, s and d indicate static and dynamic objects, k denotes the current time instance, and N denotes the number of dynamic objects in the given space.

Generally, SLAM works well under the assumption of a static environment [3]. Hence with static world assumption poses and landmark for the given measurements could be represented as,

$$p(X_k, l | Z_k) \quad (3.1)$$

When the dynamic objects are present in the environment, their states O_k and corresponding landmarks l^o are also assumed to be static by the SLAM algorithm represented in equation 3.2. This static assumption of dynamic objects affects the estimations by creating false feature matches.

$$p(X_k, l, O_k, l^o | Z_k) \quad (3.2)$$

This is one of the important limitations of SLAM that affects the quality of tracking. In highly dynamic scenarios, inconsistent feature correspondences cause tracking to even fail. Even though the camera measurement Z identifies distinctive visual features, the equation 3.2 demonstrates that pose estimation and map points of static and dynamic entities are closely intertwined. However, in the real world, the states of camera and dynamic objects are entirely independent; thus equation 3.2 can be modified as

$$p(X_k, l | O_k, l^o, Z_k) \quad (3.3)$$

The methodology implemented in stage 1 and stage 2 exploits the independent state representation to overcome the limitation of static assumption.

3.2.2. Stage 1 - Dynamic object segmentation

This stage uses the deep learning models to find potentially dynamic classes and remove corresponding features in the ORB-SLAM2. To achieve this objective, our literature review demonstrates that a single-stage instance segmentation architecture, particularly SOLOV2, could successfully generate an accurate output mask with faster inference.

We only identify classes like vehicles, pedestrians, and bicycles as potentially dynamic objects in our experiments. **The important assumption of stage 1 implementation is that regardless of how predicted classes move in their surroundings, we classify them as dynamic.** Firstly, the neural architecture of SOLOV2 is explained, and later integration of output with ORB-SLAM2 is discussed.

3.2.2.1. SOLOV2 network architecture

SOLO is a term that refers to segmenting object by locations. As implied by the name, the segmentation mask is produced depending on the image's location. The central idea is similar to that of a single-stage object detector, i.e., SOLOV2 analyzes the presence of object instances and investigates the possibility of segmenting objects based on their centre position.

SOLOV2 [49] framework splits the input image into grid cells, and each cell is responsible for pixel-wise categorization and object instance association. The architecture uses the ResNet-FPN backbone [51] to extract features at different scales from the input image, which are then utilized by the instance head and mask head. The instance

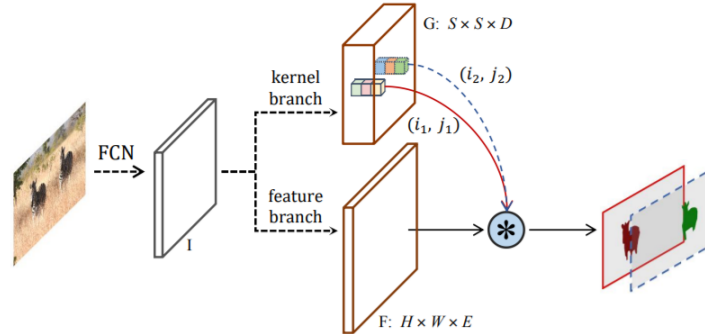


Figure 3.5: SOLOv2 mask head [49]

head is further subdivided into category head and kernel head. The category head is in charge of determining the labels for the items included inside the grids, while the kernel head calculates the filter weights for dynamic convolution. The dynamic convolution module combines the output from the mask head with the kernel head to produce the instance segmentation output. In SOLO, kernels convolve over feature maps to create a single big tensor mask which is decoupled in SOLOV2 as a separate kernel and mask to enable faster inference.

Unlike skip connection used in other segmentation architecture, the system gains translation invariance by incorporating the normalized pixel coordinates into convolutional layers through the CoordConv operator [52]. When the coordconv layer is added to the input dimension of $H * W * C$, the channel size increases by two, resulting in a dimension of $H * W * (C + 2)$.

SOLOV2 trained using conventional focal loss [53] and dice loss [54]. The details of conventional focal loss and dice loss is as follows,

Dice loss: To comprehend the dice loss, one must first analyze the intricacies of the dice coefficient. Dice coefficient is calculated as the ratio of correctly predicted pixel to the total summation of pixels contributed by prediction and ground truth which could be represented as,

$$Dice\ Coefficient = \frac{2 * \sum_i^N y_i g t_i}{\sum_i^N y_i^2 + \sum_i^N g t_i^2} \quad (3.4)$$

Dice loss is constructed based on the dice coefficient, which determines the overlap between the ground truth and predicted segmentation that helps the model learn the boundaries elegantly.

$$Dice\ Loss = 1 - \frac{2 * \sum_i^N y_i g t_i + 1}{\sum_i^N y_i^2 + \sum_i^N g t_i^2 + 1} \quad (3.5)$$

Focal loss: Focal loss is a modified version of cross-entropy loss [55] designed to work better on the highly imbalanced dataset. Focal loss contains hyperparameters that gives lower weightage to the simple examples while increasing the weight of difficult ones. Whereas cross-entropy and weighted cross entropy [56] are used to calculate the gradient flow based on the positive and negative samples only. Positive samples indicate target classes, while negative samples are usually background.

$$\text{CrossEntropy}(p_t) = -\alpha_t * \log(p_t) \quad (3.6)$$

where,

$$p_t = \left\{ \begin{array}{l} p, y = 1 \\ 1 - p, \text{ otherwise} \end{array} \right\} \quad (3.7)$$

Equation 3.6 differentiates both positive and negative samples but does not provide insight between easy and hard samples. Easy samples are those that are correctly classified as foreground or background, while hard samples are those that are misclassified. Focal loss address this problem with the following implementation,

$$\text{Focal Loss}(p_t) = -\alpha_t (1 - p_t)^\gamma \log(p_t) \quad (3.8)$$

α_t and γ are the hyper parameters that take value between 0 and 1 which are either learnt or fixed during the training. If $\gamma = 0$ then focal loss becomes cross entropy loss.

3.2.2.2. Combining dynamic object segmentation with ORB-SLAM2

This section discusses the algorithm used to blend the segmentation output from SOLOV2 with ORB-SLAM2 that helps identify the measurements that belong to potentially dynamic objects. The stage 1 implementation categorise the feature measurement Z_k into two subsets as static measurements Z_k^* , potentially dynamic object's measurement Z_k^o . These feature subset identified using segmentation mask modifies the equation 3.3 as

$$p(X_k, l|Z_k^*) \cdot p(O_k, l^o|Z_k^o) \quad (3.9)$$

Algorithm 1 describes the implementation steps involves in stage 1 to process individual images and their associated masks to split the output features into subsets. The algorithm is implemented on the feature extraction stage of ORB-SLAM2(in Frame subclass).

A key difference between our approach from other research studies is that our solution discards dynamic features during the tracking step but includes them exclusively at the mapping stage(discussed later in the chapter). Thus our implementation not only improves robustness in tracking results but also creates semantically meaningful dynamic maps. Since we preserve the details of map points of all the objects, the final result obtained by the combination of SOLOv2 with ORB-SLAM2 could be probabilistically expressed as,

$$p(X_k, l|Z_k^*) \cdot p(l^o|Z_k^o) \quad (3.10)$$

Algorithm 1 Processing features as static and potential dynamic class

```

1: for time  $i = 0$  to  $K$  do
2:    $Z_i \leftarrow RGB_i$  ⊠ ORB feature extraction
3:   for Number of  $Z_i$  do
4:     if  $Z_i$  in  $Mask_i$  then ⊠ Checking features in dynamic area
5:        $Z_i^o = True$ 
6:     else
7:        $Z_i^* = True$ 
8:     end if
9:   end for
10: end for
11: return  $Z_k^*$  &  $Z_k^o$ 

```

3.2.3. Stage 2 - Moving object segmentation

Though we adopt a state of the art instance segmentation architecture to remove dynamic objects through stage 1, the experimental results achieved during the simulation are not convincing. The main drawback comes from the inability of SOLOV2 to identify whether the predicted mask is moving or not. **The current section discusses the techniques employed for classifying dynamic objects into moving and non-moving entities.** Some conventional methods based on the geometrical approach reviewed in the literature study might well be utilized to find moving objects. We performed experiments using the geometrical approach with epi-polar constraint to detect dynamic objects in the segmentation output. But the final results obtained from geometrical methods are not consistent in all the use cases. The appendix B.1 contains the specifics of the geometric method implementation. To that extent, we started exploring the possibilities of finding the moving object using a learning-based approach for better generalization.

Action recognition [57] is an important area of computer vision that uses visual-temporal characteristics to anticipate the action in the video sequence. Using the idea from action recognition, we develop two-stream encoder-decoder CNN architecture to find the moving object from the potential dynamic subset. The inclusion of temporal features is the primary modification needed in stage 2 to detect moving subsets.

3.2.3.1. Two stream network architecture

Two-stream architecture developed in our project follows a similar structure with MODNet [58]. The model uses a RGB image and an optical flow input to detect moving objects and segment the spatial mask. The architecture contains two encoder and two decoder units. Numerous modifications have been made to the base MODNet structure to improve the final accuracy.

The encoder part of the network is based on VGG [59] backbone which extracts features from RGB and flow inputs. Optical flow input to the network is generated by FlownetV2 [60] architecture. Optical flow quantifies the temporal connection between

neighbouring frames in the near term. Moving elements observed between the two consecutive images exhibit significant differences in form and degree of flow compared to static regions. FlowNet model captures these changes as two-dimensional flow maps representing the direction of flow along the x and y axes. The flow vectors are transformed to an RGB picture based on the magnitude and direction of the flow using the colour-coded encoding [61]. This colour-coded flow image could then directly be used to identify the features using convolutional layers. The details of optical flow representation could be reviewed in section A.1.7.

The spatial and temporal features extracted by two encoders are combined in the network fusion stage. Features from intermediate layers are fused together by concatenation of channel space or by summation operation. The feature fusion helps to achieve better segmentation accuracy on moving object segmentation. The combined

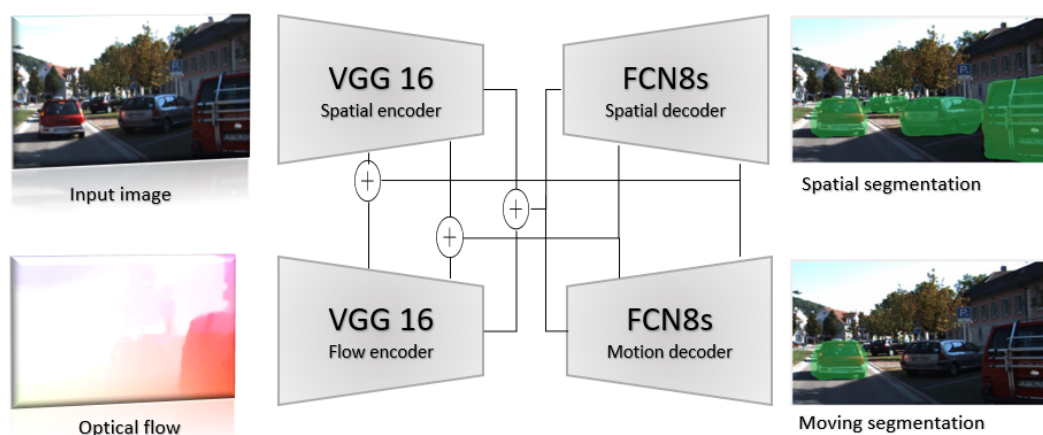


Figure 3.6: Two stream encoder decoder architecture for moving object segmentation

features are then processed by two FCN8-based [45] decoders. The spatial decoder infers the semantic segmentation of all the cars in the image, while the moving segmentation decoder infers the semantic segmentation of just the moving vehicle. ReLU and max-pooling layers used in the encoders reduce the spatial resolution of feature maps. Hence, high dimensional feature maps are upsampled by transposed convolution to obtain the required final dimension. To enhance the information flow from the encoder to the decoder at various scales, fused features from each stage are concatenated carefully into the decoder unit through skip connections, as shown in figure 5.8. This information from the encoder improves the localization of the features associated with the objects in the final segmentation output.

3.2.3.2. Combining moving object segmentation with ORB-SLAM2

Moving object segmentation categorizes the output into two distinct classes: moving and non-moving objects. These moving segmentation outputs are processed similar to that of stage 1 implementation by modifying the feature extraction stage of ORB-SLAM2. In stage 2, the extracted features are subdivided into three subsets of measurements denoted by Z_k^* , $^dZ_k^o$, $^sZ_k^o$, which represent static, moving, and non-moving dynamic features, respectively. The stage 2 implementation on the feature extraction

stage of ORB-SLAM2 is explained in algorithm 2.

Algorithm 2 Processing features as static, moving and non-moving dynamic objects

```

1: for time  $i = 0$  to  $K$  do
2:    $Z_i \leftarrow RGB_i$  ⊠ ORB feature extraction
3:   for Number of  $Z_i$  do
4:     if  $Z_i$  in  $Mask_i$  then
5:       if  $Mask_i == dynamic$  then
6:          ${}^dZ_i^o = True$  ⊠ Checking moving features
7:       else
8:          ${}^sZ_i^o = True$  ⊠ Checking non-moving features
9:       end if
10:    else
11:       $Z_i^* = True$  ⊠ Checking static features
12:    end if
13:  end for
14: end for
15: return  $Z_k^* \& {}^dZ_k^o \& {}^sZ_k^o$ 

```

Sub-categorisation of non-moving dynamic features in stage 2 overcomes the drawback of our previous implementation. The non-moving dynamic features are combined with static features to enhance the robustness of the tracking. Similar to stage 1, instead of removing moving features, we use them in semantic mapping. Combined optimisation of the subset of features could probabilistically be represented as,

$$p(X_k, l, {}^s l^o | Z_k^*, {}^s Z_k^o) \cdot p({}^d l^o | {}^d Z_k^o) \quad (3.11)$$

The first portion of the equation 3.11 is responsible for producing poses and landmarks using static and non-moving dynamic features. The second part of the equation, which detects the moving object features in the vicinity, is utilized only for mapping.

3.3. Mapping

The previous implementations in section 3.2 concentrate more on handling the dynamic objects to improve the tracking. As mentioned earlier, the robustness in tracking would inherently estimate robust three dimensional maps. However the mapping results of ORB-SLAM2 will represent all the map points as static. Inclusion of the points corresponding to the dynamic objects without modifications would also be represented as static. So, we investigate the possibility of using the semantic information obtained in stage 1 and stage 2 algorithms to enhance the quality of the map. This section discusses the technicalities involved in semantic mapping implementation and the prediction of dynamic density.

In ORB-SLAM2, a monocular camera image is processed by triangulation to transform two-dimensional visual features to three-dimensional map points. These map points

represent the extracted static visual features and do not possess any additional information about these features. Semantic mapping helps in identifying the information about individual map points generated by ORB-SLAM2. The semantic information could be merged in two ways: directly or in parallel fusion [30]. Direct semantic fusion processes the final 3D point clouds directly to understand their underlying structure and related semantics. The direct approach requires 3D annotated data and requires heavy computation to predict the structure from the point cloud. On the other hand, parallel semantic fusion utilizes the 2D image to comprehend the scene and associate the 2D semantic knowledge appropriately in 3D space. As our solution already uses deep learning models to infer semantics from images, we could immediately use the information from previous stages to fuse it in a parallel fashion. Our implementation integrates the dynamic classification and instance semantic segmentation to build a dynamic semantic map and instance semantic map. The below sub-section 3.3.1 and 3.3.2 will cover the implementation details of these semantic mapping in detail.

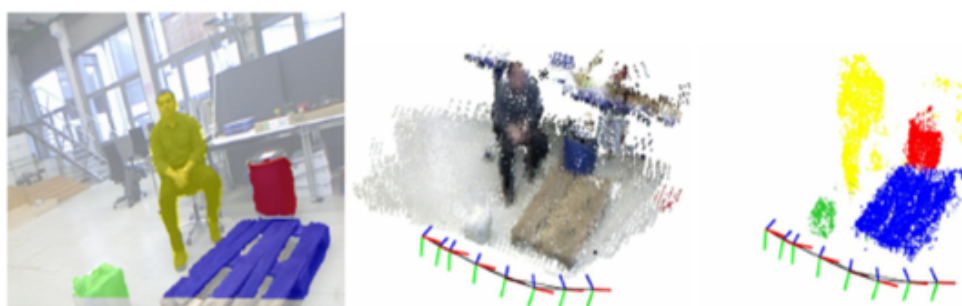


Figure 3.7: Semantic mapping. Semantic information associated with different color in 2D image(left) is associated with 3D map points(right)

3.3.1. Dynamic semantic mapping

In general, semantic mapping associates the object class present in the environment such as pedestrians and cars with map points. We take the idea of semantic mapping one step ahead where in addition to finding the car present in the 3D space, we associate the dynamic information. Hence, in dynamic semantic mapping, the details of the dynamic subset obtained in the feature extraction stage are used in the mapping thread. 2D-3D data association from feature extraction to mapping thread is the key to translating image semantics to three-dimensional space.

The algorithm 3 details the various steps involved in the data association to build a dynamic semantic map based on the output of stage 2. It is important to note that the subset of feature measurements varies as per the stages. The algorithm 3 could be slightly changed to adapt the result from stage 1.

Stage 1 classifies static or potentially dynamic features, producing static landmark l and a dynamic landmark l^o respectively. While stage 2 classifies them into three subsets creating static landmark l , non-moving dynamic landmark $^{sl^o}$ and moving points $^{dl^o}$. Thus final map generated by stage 2 output provides dynamically separable semantic maps in three-dimensional space.

Algorithm 3 Dynamic semantic mapping

```

1: for  $Z_i = 0$  to  $K$  do
2:    $points \leftarrow Triangulate(Z_i, Z_i + 1)$  ⊠ Depth estimation
3:   for Number of points do
4:     if  $Z_i^* = True$  then ⊠ Static points
5:        $l = True$ 
6:     else if  ${}^sZ_i^o = True$  then ⊠ Non-moving dynamic points
7:        ${}^sl^o = True$ 
8:     else if  ${}^dZ_i^o = True$  then ⊠ moving dynamic points
9:        ${}^dl^o = True$ 
10:    end if
11:  end for
12: end for
13: return  $l$  &  ${}^sl^o$  &  ${}^dl^o$ 

```

3.3.2. Instance semantic mapping

The implementation of SOLOV2 in our pipeline helps us to take advantage of instance segmentation output to build instance semantic maps [62] [63]. While the SOLOV2 algorithm provides instance segmentation for each image, the output mask cannot be correlated across successive frames. Thus, it is critical to track the instances produced between two consecutive frames in order to construct instance semantic mapping. A tracking module is developed to track instances between consecutive images that generate an instance id for each image that could be later used in mapping to identify two similar objects across frames.

3.3.2.1. Multi-object tracking

The tracking module is developed based on the idea of multi-object tracking (MOT) [64]. In general, object tracking examines the video to find the existence of the same object seen in the previous frame and tracks them until the object becomes apparent. Tracking by detection is a common technique adapted for addressing MOT problems in which the output of detection is acquired first, and tracking is done afterwards. After the detection, the predicted area is cropped as patches from the corresponding RGB image. A separate CNN model subsequently processes these image patches to produce feature embedding. Feature similarity among the acquired embeddings on two image frames is used to track the same object by assigning it to a unique instance-id. The discussed approach is also known as the Separate Detection and Embedding (SDE) model [66]. The other techniques include joint detection embedding and cropping patch directly from feature maps. We chose to adopt the separate detection and embedding technique in our study because it enables the tracking algorithm to be implemented separately without any modification to SOLOV2. The independence of our implementation allow us to adapt pre-trained weights of SOLOV2 and even extend the tracking module to work with different architecture in future.

The feature embedder model is the principal part of the tracking module. Any pre-

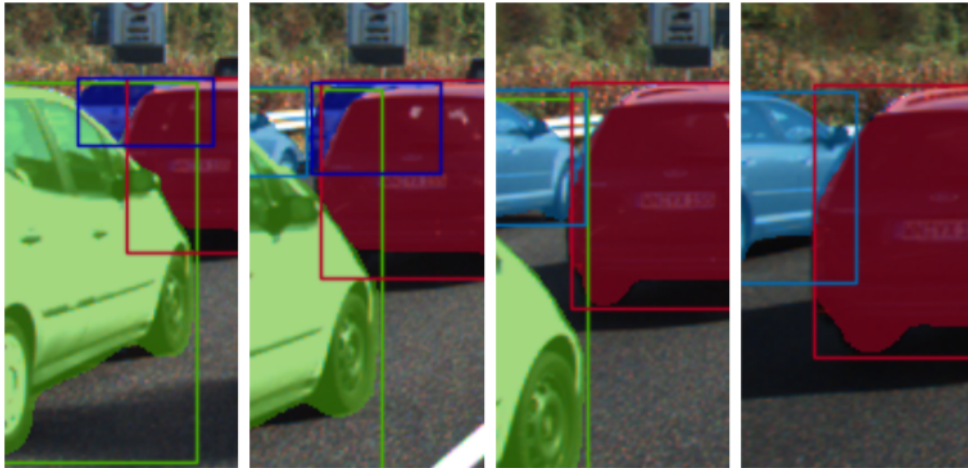


Figure 3.8: Multi-object tracking [65]

trained CNN based backbone architecture would be appropriate for the feature embedding model. However, ResNet18 architecture [67] pre-trained on the ImageNet dataset is chosen as the feature embedder in our pipeline. In addition to the feature embedding model, our tracking module has a class constraint and IOU filtering, which help us to reduce the computation needed in the feature matching stage. The section 5.1.3.1 will demonstrate why ResNet18 was selected and overall implementation of tracking module in detail.

3.3.2.2. Combining multi-object tracking with ORB-SLAM2

Each segmentation mask of SOLOV2 is processed in the tracking model possesses an associative instance identification number(id). The results of tracked instances are stored as an image(say as track image) by representing the pixel value of the segmentation mask replaced with instance id value as shown in figure 3.9. In addition to the mask produced by SOLOV2, the ORB-SLAM2 algorithm is now adjusted to receive track images as well. As a result of this modification, ORB-SLAM2 now accepts RGB images, segmentation mask images, and track images.

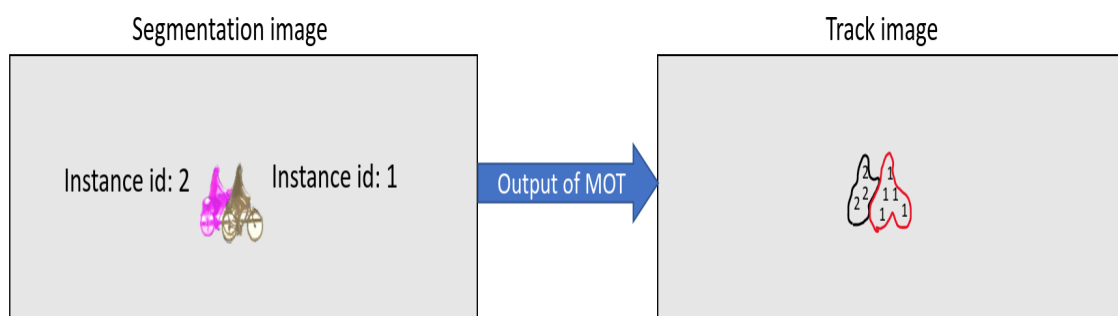


Figure 3.9: Track image representation

The fundamental step in the instance semantic mapping is to identify a cluster of points belonging to the same object instance and associate them with unique semantic labels. The algorithm 4 describes how tracked identifiers from object tracking modules

could be correlated with three-dimensional points in order to identify the individual cluster.

Algorithm 4 Instance ID association

```

1: for  $Z_i = 0$  to  $K$  do
2:    $points \leftarrow Triangulate(Z_i, Z_i + 1)$  ⊠ Depth estimation
3:    $instance_{id} \leftarrow Z_i$  at  $track_i$  ⊠ Object ID from tracking module
4:   for Number of points do
5:     if  $Z_i^o = True$  then ⊠ 3D association
6:        $l_{instance}^o = instance_{id}$ 
7:     end if
8:   end for
9: end for
10: return  $l_{instance}^o$ 

```

The representation of track id as track image helped us to directly extend the implementation on the feature extraction module done on stage 1 and stage 2. The 2D-3D data association involves linking instance ids from the track image to each feature measurement (map points are triangulation of similar features) if the extracted features fall in one of the contours on the track image. For instance mapping, we only consider all landmark points that belong to potential dynamic objects $l_{instance}^o$ (results of stage 1) and specifically on car class.

3.3.3. Dynamic density estimation

This section explains the utilization of the output of semantic maps through dynamic density estimation. The dynamic density is nothing more than a metric for enumerating the degree of dynamic interaction occurring in the local environment. Identifying the level of dynamic interaction in the environment enables the robots' navigation to function more effectively by establishing appropriate priors to regulate the speed.

In contrast to tracking and mapping, dynamic density is calculated instantly based on the current location of the ego-vehicle and is therefore shown only as visualization output at the present instant. There are two approaches for estimating dynamic interaction: one utilizes two-dimensional semantic information directly received from the neural network, and the other is based on three-dimensional semantic maps. Though two-dimensional semantics is used in the previous mapping implementation, it is ineffective in dynamic density estimation as the results become more reliant on the image being seen. Due to the limitation in the camera focus, the dynamic objects that are not recorded in the field of view could miss few objects present around it. In contrast, the second approach that utilizes generated maps overcome the limitation, providing more intuitive information around the space of the ego vehicle.

The dynamic semantic map from stage 2 encapsulates all the information about the moving dynamic objects which is used to identify the dynamic classes that move in a

Algorithm 5 Dynamic density estimation

```

1: dynamic_count = 0
2: for Number of points in map do
3:   if  $d_l^o = True$  then
4:      $x, y, z \leftarrow d_l^o$ 
5:      $x_c, y_c, z_c \leftarrow X_k$ 
6:     if Equation 3.12 is satisfied then
7:       dynamic_count ++
8:     end if
9:   end if
10: end for
11: if dynamic_count  $\geq 10$  then
12:   display = highly dynamic
13: else
14:   display = less dynamic
15: end if

```

local 3D space. With the centre set on the current camera pose, the spherical constraint determines if the dynamic points are bounded inside the sphere. Spherical constraint to find the 3D local space is given by,

$$\begin{aligned}
 x, y, z &= d_l^o \\
 x_c, y_c, z_c &= X_k \\
 \sqrt{(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2} &\leq r
 \end{aligned} \tag{3.12}$$

where:

- d_l^o = landmark associated with moving class
- X_k = current camera pose
- r = radius of sphere

The map points satisfying the bounding conditions are the only points that involve in the density calculation. The algorithm 5 explains how local space is classified into high or low dynamic interaction states. To account for the potential outliers in the mapping stage, we consider the local area to be dynamic only if at least ten dynamic points satisfy the equation 3.12.

4

Dataset and metrics

4.1. Dataset

Proper choice of the dataset is essential to test our implementations proposed in the methodology. This section deals with the selection of our dataset for testing SLAM and segmentation models. Furthermore, it covers our contribution on generating a new dataset for moving object segmentation.

4.1.1. SLAM

To validate our proposed methodology on ORB-SLAM2, we need an odometry evaluation dataset specifically tailored for visual SLAM. We conducted a review on three widely used datasets, namely the EuRoC [68], TUM [69], and KITTI [70]. EuRoC dataset is a collection of stereo sequences of the indoor environment recorded using a micro aerial vehicle. Also, the TUM dataset contains collection of sequences that are primarily focused on the indoor environment. On the other hand, the KITTI odometry dataset gathers outdoor environments, especially urban and highway settings. Outdoor spaces have a high level of dynamic interactions, which aligns our interest in the KITTI dataset.

KITTI odometry dataset contains 22 sequences, out of which only 11 sequences have ground-truth information. In the 11 sequences only for six sequences loop closure is possible as shown in the table 4.1. After manually observing these six sequences, we noticed that only sequences 05 and 07 include a mix of moving and stationary vehicles, which could be ideal to evaluate our approaches. However, to better understand our methodology in a highly dynamic setting, we also included the 04 sequence in our list of selected datasets.

The ground truth of sequence 04 has 271 poses where camera motion runs almost straight with path length closer to 394 meters. Sequence 07 has 1101 ground-truth poses covering the loop with an average length of 694 meters, and sequence 05 is the longest path with a stretch of 2205 meters with 2761 ground-truth poses.

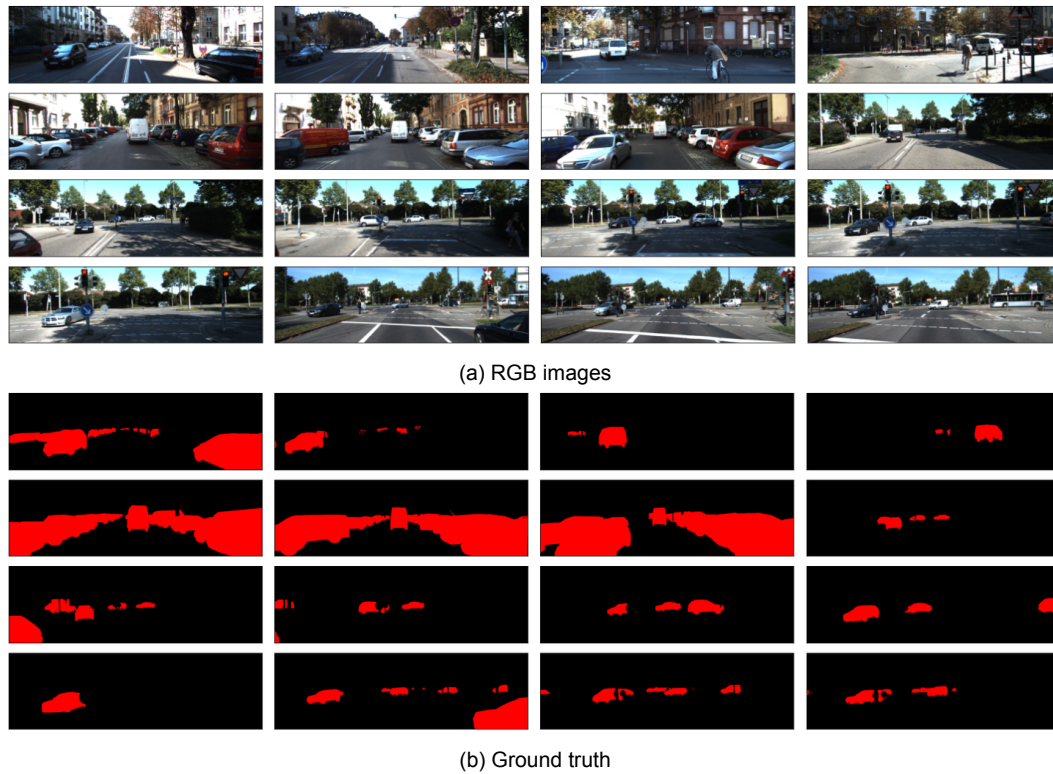


Figure 4.2: KITTI segmentation dataset

4.1.3. Stage 2 - Moving object dataset

The two stream neural architecture used in stage 2 requires details of the movement of cars and optical flow images and RGB images to train the model. These requirements are satisfied only by the KITTI MoSeg dataset [72] which contains semantic segmentation for all the cars and the bounding boxes for moving cars detection along with optical flow images. Since our implementation requires ground truth segmentation input for both the decoders, the KITTIMoseg dataset needs to be modified to generate a segmentation mask for the moving object from the bounding box. In addition, flow images available in the dataset are distorted, representing inaccurate object boundaries. As the authors do not disclose the details of the generation of flow images in the paper, we use FlowNetV2 to regenerate the flow images.

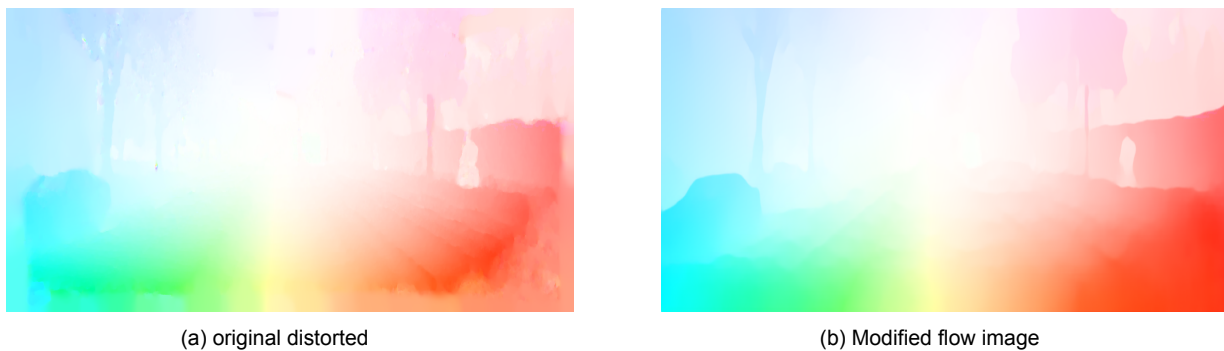


Figure 4.3: Improving boundaries of flow image KITTIMoseg

4.1.3.1. New dataset generation

In this project, we created a new dataset based on KITTI odometry sequences targeting moving object segmentation. Polygon tool Labelme [73] is used for annotation. The polygon points are converted into segmentation images with a separate python script. In total, 3085 static cars and 893 moving cars are annotated. FlowNetV2 architecture is used for generating optical flow images. The new dataset contains 1014 distinct sample frames with associated flow images and mask. A sample of our newly created dataset is shown in the figure 4.4.

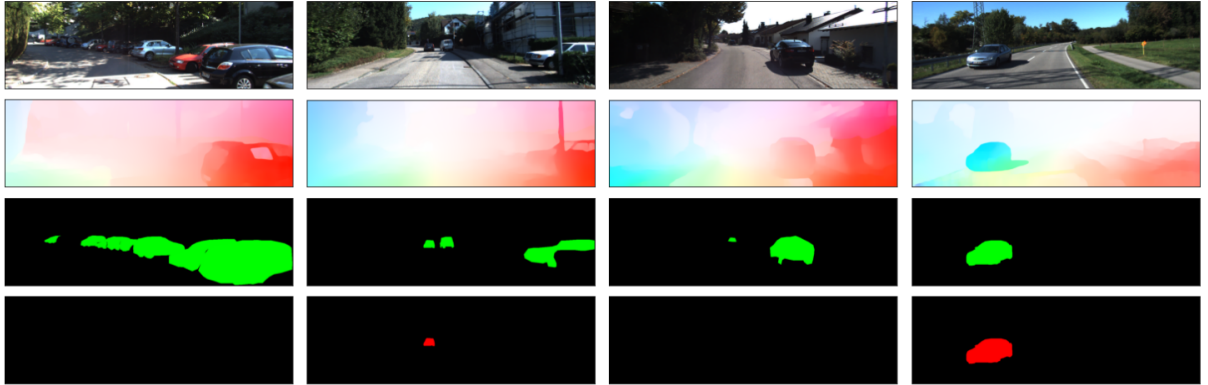


Figure 4.4: Stage 2 dataset created based on KITTI dataset. Row (i) RGB image, (ii) Flow image, (iii) Segmentation mask of all cars, (iv) Segmentation of moving cars

4.2. Metrics

4.2.1. Absolute trajectory error - SLAM

This section explores the metrics used to compare the SLAM tracking results obtained in each stage of our methodology with baseline. Quantitative metrics such as absolute trajectory error and relative pose error are used to evaluate the estimated trajectory against the ground truth. **Absolute trajectory error** used in ORB-SLAM2 paper is utilized in our experiment to facilitate comparisons. Absolute trajectory error first aligns

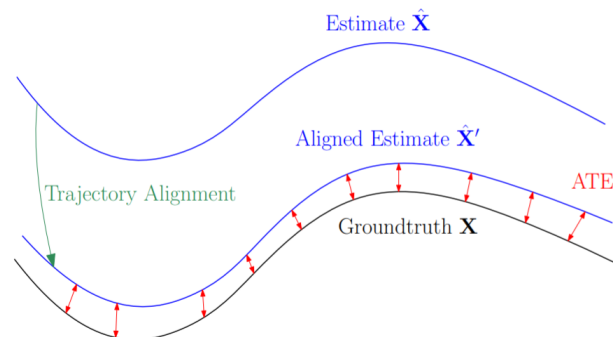


Figure 4.5: Trajectory alignment for absolute trajectory error [74]

the estimated trajectory with ground truth and then calculates the root mean square

error between aligned trajectory and ground truth [75]. Due to the scale ambiguity in monocular slam, alignment of the trajectory becomes mandatory to obtain accurate estimates. The Umeyama method [74] is used to estimate the alignment transformation as a least square problem, as illustrated in the algorithm 6. The estimated transformation for rotation, translation and scale based on the Umeyama method would result in the trajectory alignment as shown in the figure 4.5.

Algorithm 6 Umeyama - Alignment transformation estimation

- 1: **Input:** Estimated positions \hat{P} and ground truth position P
 - 2: Calculate the **mean** — $\mu_{est} = \frac{1}{N} \sum_{n=0}^N \hat{P}_n$ and $\mu_{gt} = \frac{1}{N} \sum_{n=0}^N P_n$
 - 3: Calculate the **variance** — $\sigma_{est}^2 = \frac{1}{N} \sum_{n=0}^N \|\hat{P}_n - \mu_{est}\|^2$ and $\sigma_{gt}^2 = \frac{1}{N} \sum_{n=0}^N \|P_n - \mu_{gt}\|^2$
 - 4: Calculate the **co-variance matrix** — $\Sigma = \frac{1}{N} \sum_{n=0}^N (P_n - \mu_{gt})(\hat{P}_n - \mu_{est})^T$
 - 5: Perform singular value decomposition — $\Sigma = UDV^T$
 - 6: **if** $|U| |V| < 0$ **then**
 - 7: $W = \text{diag}(1, 1, -1)$
 - 8: **else**
 - 9: $W = \text{Identity Matrix}$
 - 10: **end if**
 - 11: Calculate **rotation** $R = UWV^T$
 - 12: Calculate **scale** $s = \frac{1}{\sigma_{\hat{p}}^2} \text{trace}(DW)$
 - 13: Calculate **translation** $\mathbf{t} = \mu_{gt} - sR\mu_{est}$
 - 14: **return** R, s, t
-

After the trajectory alignment, the root mean square error of the euclidean distance between new estimated poses and ground truth poses obtains the absolute trajectory error. The primary advantage of the metrics is that a single number captures overall variation in position estimation over the whole trajectory. However, the calculated error is sensitive to time. During experiments, ATE for the change in the position is calculated by the following formula,

$$\text{ATE}_{\text{pos}} = \sqrt{\left(\frac{1}{N} \sum_{n=0}^N \|\mathbf{p}_n^{\text{aligned}} - \mathbf{p}_n^{\text{gt}}\|^2 \right)} \quad (4.1)$$

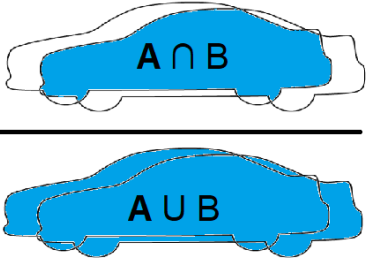
where:

p = position of individual pose

4.2.2. IOU estimation - Segmentation

Average precision is used as a metric to evaluate the accuracy of segmentation. Average precision is usually calculated over all the classes and across all IOU threshold. IOU [76], also known as intersection over union, determines the overlap of the ground

truth(A) and predicted masks(B) to distinguish true positives, false positives, and false negatives. If the IOU value is higher than the set threshold, it is considered as correctly predicted. Usually, IOU threshold values are iterated from 0 to 1 in steps of 0.1, and their final average is used for the comparison.

$$\text{IOU} = \frac{\text{Overlap area}}{\text{Union area}}$$


The diagram illustrates the components of the IOU metric. It shows two overlapping blue shapes representing sets A and B. The intersection of the two shapes is shaded and labeled $A \cap B$. The union of the two shapes is shaded and labeled $A \cup B$. The equation $\text{IOU} = \frac{\text{Overlap area}}{\text{Union area}}$ is shown to the left of the shapes.

Figure 4.6: IOU metrics

5

Experimentation and results

The chapter covers various experiments conducted in the project and discuss the obtained result by comparing against the baseline. Though the final objective is to build robust mapping by improving the SLAM process through the learning-based method, we first analyse the deep learning architectures separately and then investigate the influence of the model on SLAM in the second section.

5.1. Object segmentation and tracking models

Different convolutional neural network models discussed in the methodology are implemented to improve the quality of the SLAM. This section explains the experiment details of all the model architectures used in the pipeline and their limitations.

5.1.1. SOLOV2 - Dynamic object segmentation

The choice of single-stage instance segmentation architecture SOLOV2 for dynamic object removal adapted in development stage 1 [3.2.2.1] is justified by comparing the segmentation accuracy with other popular baseline models used in SLAM. Based on a literature review, it is apparent that Mask-RCNN is widely used, for instance segmentation architecture among SLAM researchers. Hence, we evaluate the effectiveness of our proposed single-stage architecture SOLOV2 by comparing it to the two-stage instance segmentation architecture Mask-RCNN. As explained in the previous chapter, the KITTI segmentation evaluation dataset is used in our accuracy comparison. Given that the primary dynamic class of the KITTI sequence is the car, we disregard all other classes for evaluation.

Initially, SOLOV2 was built from scratch in the Pytorch framework. Since our specified classes are pedestrians and cars, we adopt the pre-trained model proposed in the paper. However, the adoption of pre-trained COCO weights directly to our implementation became infeasible due to changes in model parameter naming convention. Hence SOLOV2 is later implemented in Python using detectron2 [77], a vision library developed by Facebook AI researchers using Pytorch framework based on the reference provided in the paper. The post-processing steps in SOLOV2 have been

tweaked to infer just the vehicle labels. For comparison, Mask-RCNN is also implemented in Python. Similar to SOLOV2, pre-trained weights on the COCO dataset are used in Mask-RCNN, and the post-processing stage of Mask-RCNN is modified to infer car segments.

5.1.1.1. Result and discussion

Segmentation output for both the models is generated for 200 KITTI images with a batch size of one and a confidence score of 0.3. Models are tested in the Intel i7-8850H 16GB RAM CPU system configuration and NVIDIA Quadro P1000 GPU.

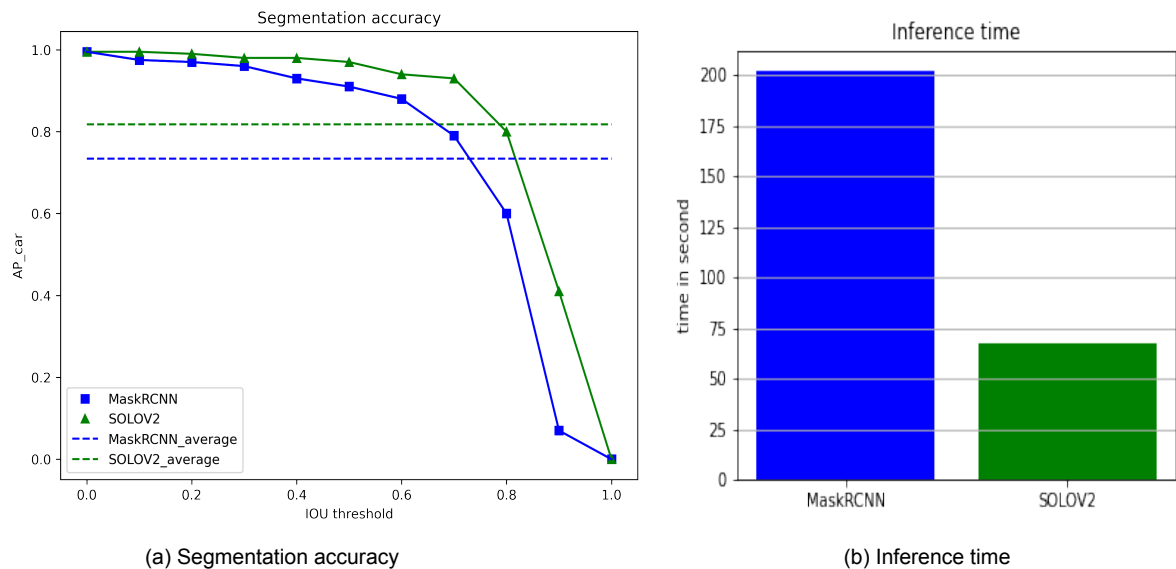


Figure 5.1: Quantitative comparison - SOLOV2 and Mask-RCNN

The quantitative result from the figure 5.1 shows that SOLOV2 improves the segmentation accuracy by 10.27% compared to Mask-RCNN. Mask-RCNN takes about 201.96 seconds to process 200 images, while SOLOV2 takes just 67.25 seconds, i.e., SOLOV2 takes **0.4 seconds** to process a single image, whereas two-stage architecture takes **0.91 seconds**. Overall, the proposed single-stage architecture have 2.5 times faster inference time. The faster inference is analogous to the fact that, unlike Mask-RCNN, which predicts the bounding box first and then segments inside it, SOLOV2 predicts the instance mask directly as a classification problem.

Based on the quantitative analysis, it is evident that MaskRCNN misses most of the cars when placed closer to each other, while SOLOV2 captures the accurate boundaries of all the available instances. One such example is shown in figure 5.2. Mask-RCNN detects the objects in a fixed resolution which results in poor accuracy on complex scenes. The quantitative and qualitative evaluations justify our choice of the single-stage instance segmentation architecture in terms of speed and accuracy.



(a) SOLOv2



(b) Mask-RCNN

Figure 5.2: Qualitative comparison of SOLOv2 and Mask-RCNN

5.1.2. Two stream architecture - Moving object segmentation

This section covers the basic implementation details of the model used in the stage 2 module [3.2.3.1]. The neural network architecture is developed in python using the Pytorch framework. VGG-16 architecture imported from torch models and initialised with pre-trained weights based on ImageNet. Instead of just employing binary cross-entropy as recommended in the base MODNet paper, dice loss is implemented in combination with binary cross-entropy for training the model. The details of dice loss

Parameter Name	Values
Backbone	VGG-16
Optimiser	Adam
Loss function	BCE and dice loss
Learning rate	1e-4
Learning decay step	10
Learning decay coefficient	0.1

Table 5.1: Parameter setting - Motion segmentation model

and cross-entropy is discussed in the section 3.2.2.1. Most of the hyperparameters are maintained consistent with the MODNet implementation. Model is trained using Adam optimiser [A.2.1.4] with a batch size of 1. The current model does not incorporate any form of regularisation. By default, VGG-16 is trained with an input resolution of 224×224. However, during our experimentation, we increase the input resolution as 448×448 and 370×1226 to improve segmentation accuracy based on the research paper [78]. The table 5.1 shows the list of parameters used during training. The intermediate feature fusion, pre-trained weight initialization and joint optimization of

two decoder units accelerates the convergence of the loss function.

5.1.2.1. Result and discussion

Initially, we experiment on the implementation of intermediate features with channel concatenation and summation junction. The final segmentation results were found to be better with summation junction. Loss functions have a drastic influence on the final accuracy. The loss function for the final model is selected by experimenting with binary cross-entropy and dice loss. Each trial model is trained with an input resolution of 224×224 for 60 epochs. The result of the convergence of each loss function reveals dice loss is better. The reason is that dice loss works pretty well with dataset class imbalance (with respect to background) that occurs on segmentation. In our dataset, the moving and non-moving mask are not proportionate, and multi-class problem could make the dice loss convergence unstable. So in the rest of the experiment, we train our model using the combined loss of dice loss and binary cross-entropy with lambda as a hyper-parameter to switch between the losses.

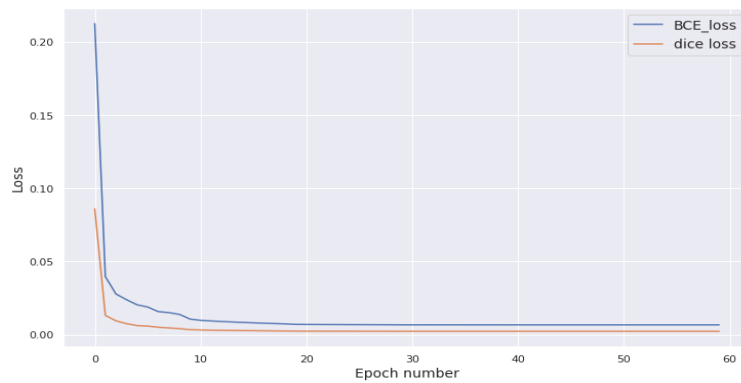


Figure 5.3: Loss function BCE loss vs dice loss

Finally, the input resolution of 448×448 and 370×1226 is used in training to see the trade-off between the accuracy and inference time improvement. Based on the above experimental result, our proposed model trained with the combined loss for input resolution of 370×1226 achieves the best outcome. We chose to train with 30 epoch, as the loss during the experiments always converge after 25 epoch. Our final model only attains 35% mIOU as the segmentation mask of moving objects is not entirely accurate.



Figure 5.4: Moving object segmentation - Row (i) RGB image (ii) Ground truth (iii) segmentation result

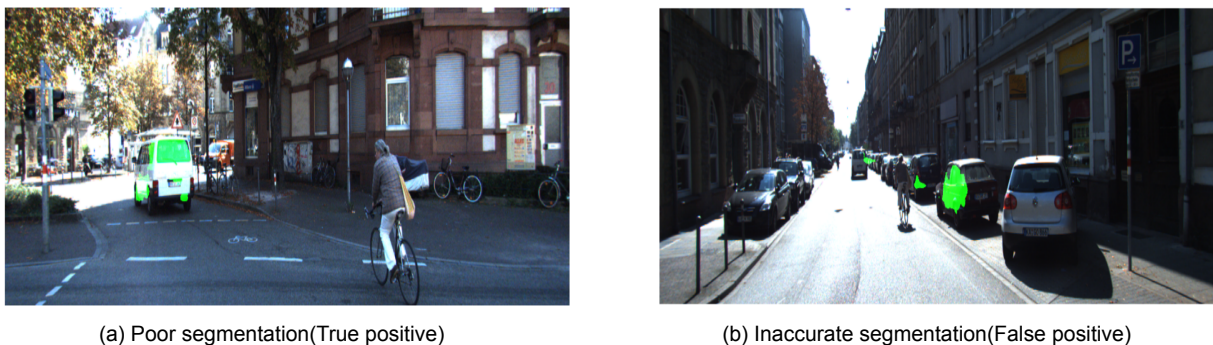


Figure 5.5: Moving object segmentation - Analysis

Segmentation results of moving objects obtained from the model fall short of our expectations. Semantic segmentation mask based on spatial features from decoder 1 yields good results by consistently predicting all the cars in the scene. This helps us infer that the model can learn spatial features but does not learn temporal features correctly. The inference is substantiated with the following discussion.

The model works considerably well when the relative change between the images is purely translational. However, the main problem in a pure translational situation is that FlowNet does not capture certain vehicles observed in far depth. In such a scenario, our model learns the spatial cues assuming the vehicle appears in the centre to be moving. Mis-classification occurs on the object present at the centre is due to the influence of spatial features over temporal features.

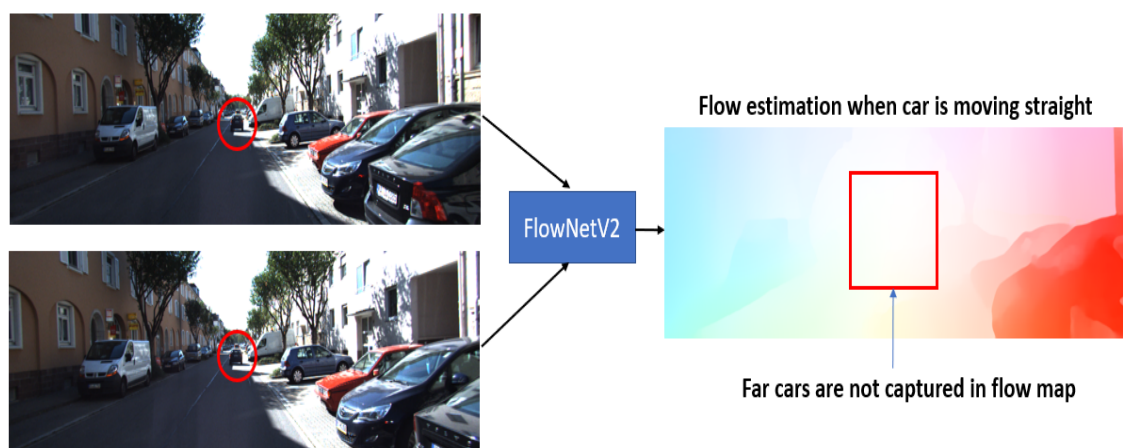


Figure 5.6: Flow image on pure translation

The observational study done on the dataset reveals that most of the moving cars lie at the centre of the image, which might bias the model. The heat map 5.7 representing the concentration of moving objects based on 2000 training images supports our discussion.

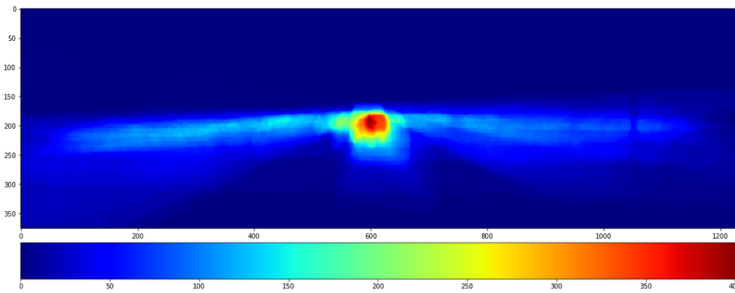


Figure 5.7: Heat map of moving object in the dataset

On closer examination of the behaviour, it is apparent that the output of FlowNet plays a crucial role in determining the final results. The flow generated by the FlowNet model is very accurate only when there is no relative transformation occurs between two images in the sequence, i.e., when there is no ego-motion. So the accuracy of moving object segmentation on the static camera is relatively high.

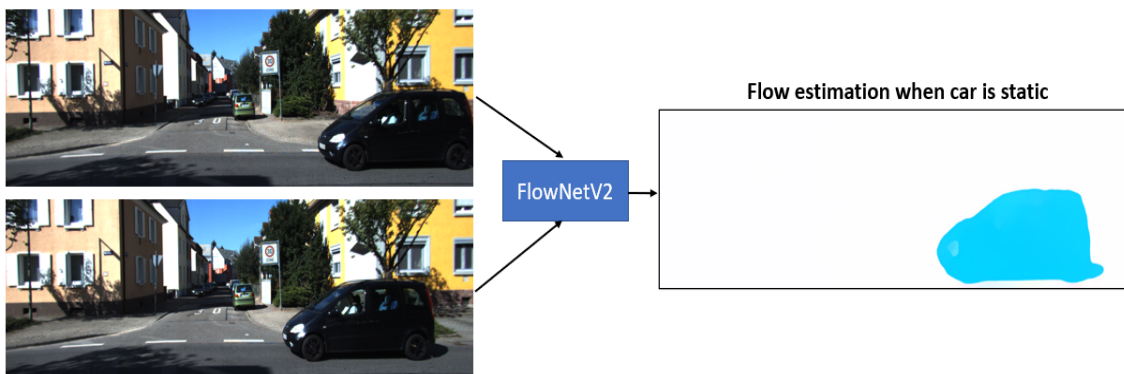


Figure 5.8: Flow image without camera motion

On the other hand, if the relative transformation between image sequences is solely rotational, the resulting flow does not offer enough information to comprehend the real-world changes. Figure 5.9 shows the inability of the FlowNet model to estimate the flow image on the KITTI sequence when the car is turning left.

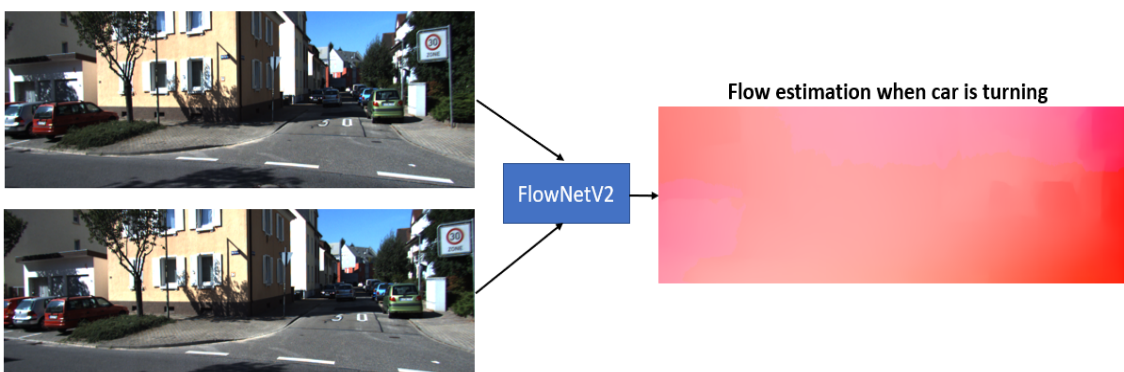


Figure 5.9: Flow image on pure rotation

Optical flow estimated by FlowNetv2 depends exclusively on the image quality, hence performs poorly on areas with shadow, reflection, and illumination changes [79]. The reason for the sub-optimal results is primarily due to FlowNet's inability to fragment the ego-motion flow. We need a precise depth estimate to have an appropriate ego-motion estimation. As we use a monocular set-up for our project, the depth estimation is uncertain. Depth estimation using monocular cameras is a separate domain to explore. Current research on moving object recognition with a camera utilizes other sensor modalities to compute depth and suppress ego-motion in order to get an accurate estimate of optical flow. A recent study about moving object estimation in SMSNet [80] and VM-Modnet [81] reveals that odometry data and IMU are used for ego-motion compensation. Methods for improving the results of moving object segmentation are discussed in our future study. Currently, to circumvent the problem of incorrect shape estimation of moving objects, the output of moving object segmentation is combined with SOLOV2 based on IOU. The output of combined segmentation is shown below,

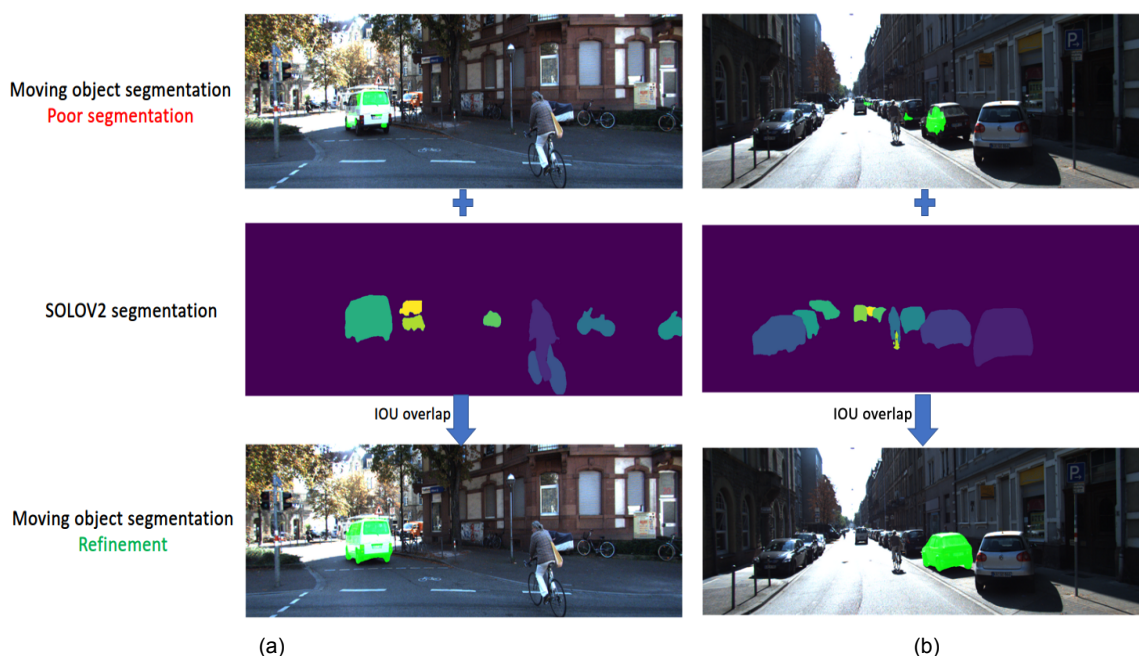


Figure 5.10: Moving object segmentation - Refinement

5.1.3. Multi-object tracking

The multi-object tracking module of the segmentation output used in mapping [3.3.2.1] is implemented in a python programming language. For each output mask produced by SOLOV2, the corresponding segmentation from the RGB image is cropped. The patch would then be re-scaled to fit into the ResNet18 model, which then generates feature embedding.

Our implementation of class constraint allows only the subset of features that belongs to the same class to be used in feature matching. These feature embedding after the class constraint is filtered using IOU metrics (details of IOU metrics is discussed in section 4.2.2). IOU filtering is hinged on the assumption that the relative velocity be-

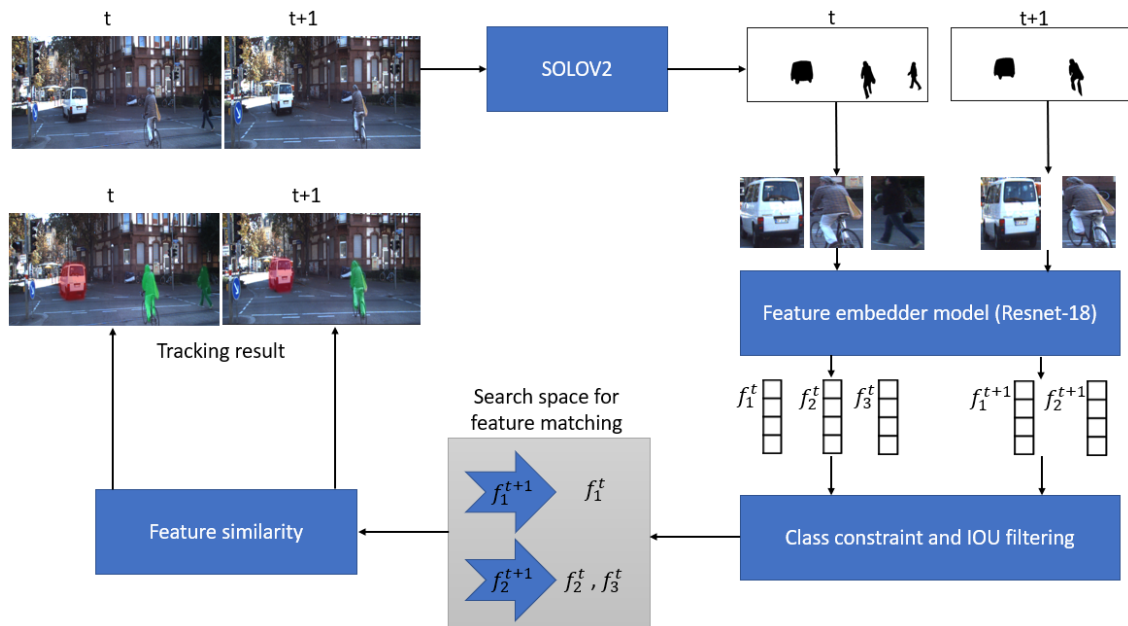


Figure 5.11: Implementation of tracking module

tween two frames does not vary abruptly. In other words, the relative transformation of object states between two sequences is always bounded. This assumption facilitates the IOU filtering to find the overlapping mask instances in two image frames. Only masks overlapping with previous segmentation output are used in the feature similarity stage, which increases tracking accuracy by eliminating false positives and reducing computation time. The Euclidean distance between two embedding vectors is used to compute feature similarity in which identical items get a lower similarity score. Figure 5.11 depicts our pipeline amended in the tracking module.

5.1.3.1. Selection of feature embedding model

This section justifies the choice of ResNet-18 architecture as the feature embedding model. The objective of the experiment is to find the architecture with reduced model complexity that helps us to achieve faster inference. We select popular backbone architectures such as VGG-16 [59], AlexNet [82], Resnet-18 [67], DenseNet-161 [83] and MobileNetV2 [84] for our experimentation.

The pre-trained backbone models from the PyTorch framework are directly used in the experiments. Input resolution of 224×224 is used in all of our experiments. From the graph 5.12, one could conclude that AlexNet outperforms all other models in terms of inference time, while MobileNetV2 has a lower memory requirement. Careful observation for a trade-off between inference time and memory shows that ResNet-18 has an inference time of 45ms, which is 45% higher than AlexNet but it requires 80% lesser space on the upside. Similarly, when compared to MobileNetV2, ResNet-18 provides 32% faster inference at the cost of a 30MB additional memory requirement. From these trade-offs, it is evident that ResNet-18 would be ideal as the feature embedding model.

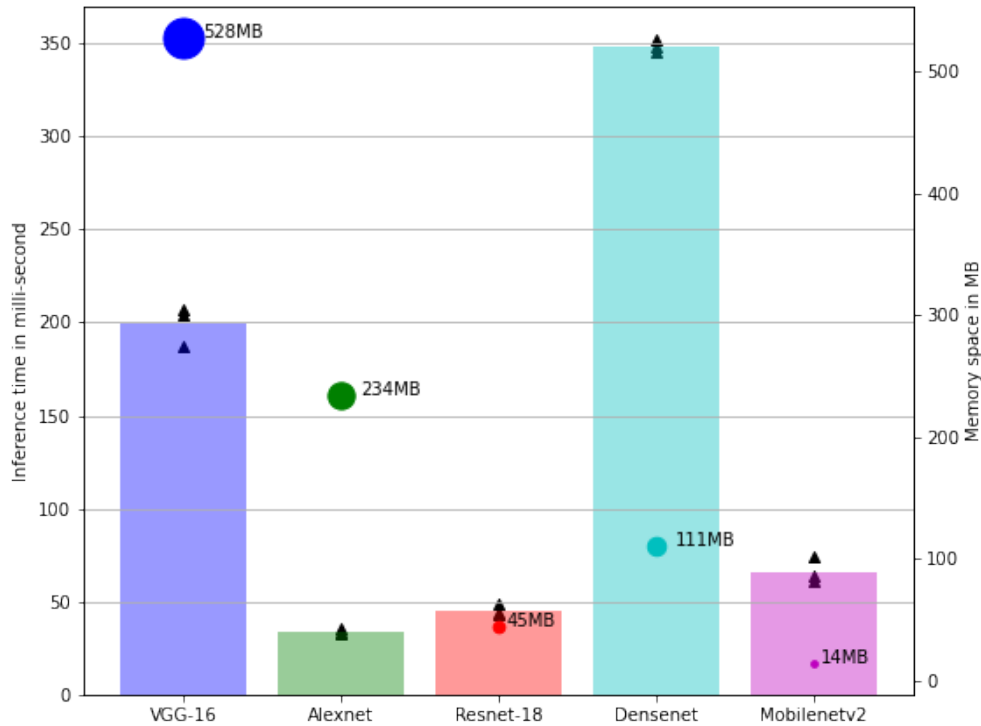


Figure 5.12: Trade-off experiment between inference time and memory for embedder model

5.1.3.2. Result and discussion

The final results of multi-object tracking are validated qualitatively in outdoor scenarios. The result of object tracking experimented on the KITTI sequence is shown in the figure 5.13. The same colour observed two image frames represent the same object present in the scene. For correlating the track ids across the whole sequence, the pixel value of the segmentation output is replaced with the identification number as discussed in 3.3.2.2. The current implementation of the tracking module only checks the correlation of the segmentation mask between two consecutive images. In case of occlusion or missed detection due to poor confidence, the object is reassigned to a new identification number. This limitation prevents us from comparing our implementation quantitatively with other studies.

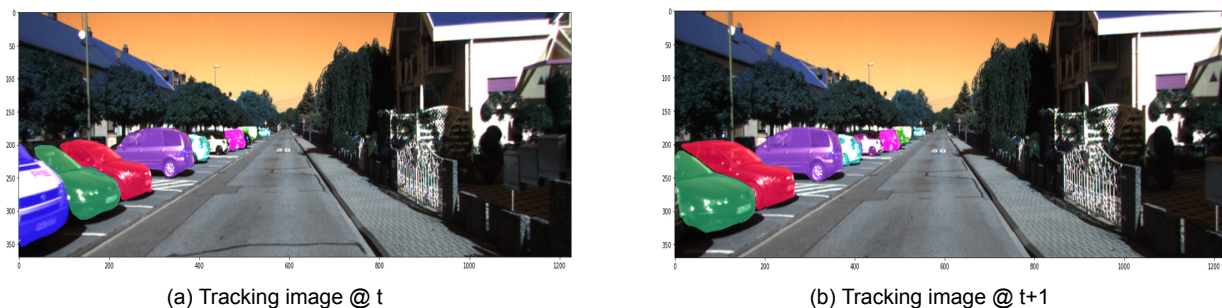


Figure 5.13: Multi-object tracking - Result

5.2. Experiments on SLAM

Robotics Operating System facilitates communication among various implemented modules. The primary benefit of utilizing the ROS platform in our experiment is that it connects various applications written in languages like C++ and Python. In our case, SLAM was implemented in C++, and deep learning models in python, making ROS an ideal platform to complete the pipeline. Individual modules as ROS Node might publish or subscribe to messages as particular ROS topics. A ROS topic is a messaging channel. Each ROS message follows a unique data structure that could be used from preset libraries or customized. Based on the methodology and experiments on neural architecture discussed in the previous section, it is evident that the ORB-SLAM2 package needs to be modified to subscribe segmentation output and tracking instance-id along with monocular image input. The corresponding ROS topics are indicated as `/camera/usb_cam_1/image_raw`, `/mask_image`, `/track_image` respectively.

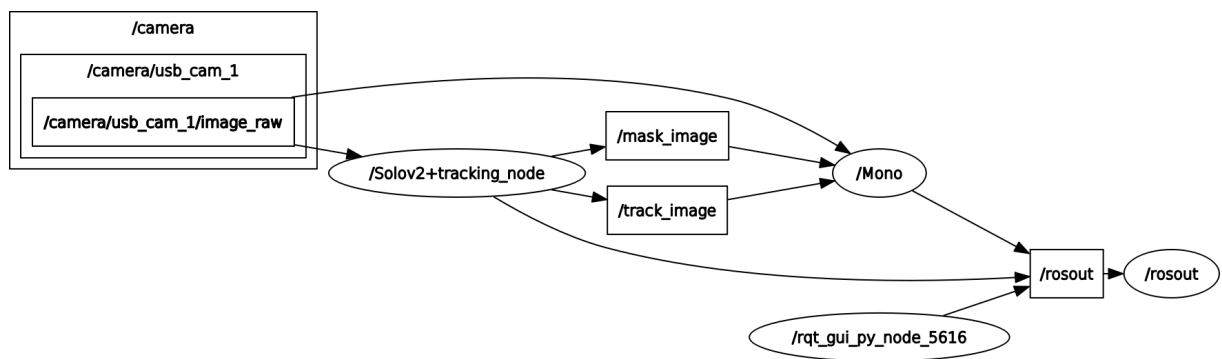


Figure 5.14: ROS implementation of our pipeline

`/Solov2+tracking_node` combines the single-stage instance segmentation model with multi-object tracking that publishes the mask and track results. ROS graph for the stage 1 implementation with tracking module is shown in figure 5.14. The same could be extended to stage 2 by the addition of the motion segmentation model. To minimize latency during SLAM operations, all relevant images are usually recorded and played from ROS bags.

5.2.1. Tracking

This section focuses on improving the tracking quality by removing the dynamic interaction. We compare the odometry results obtained from ORB-SLAM2 by integrating different development stages and discuss the limitation of individual stages separately.

5.2.1.1. Base - ORB-SLAM2

Our initial experiments concentrate on creating the baseline results using ORB-SLAM2 [3.2.1] without integrating modules from stage 1 and stage 2. We directly used the absolute trajectory error results from ORB-SLAM and DynaSLAM study as the baseline for comparing three KITTI sequences 04,05, and 07. DynaSLAM adapts Mask-RCNN architecture to remove the dynamic objects from the surroundings, which helps us to

validate our SOLOV2 implementation on SLAM. The average of the root mean square of the absolute trajectory (RMSE) error [75] after five executions in each of the three sequences is used as the final result to establish consistency. The main factors that affect the consistency of final results are initialization and the latency introduced by parallel computation. For appropriate comparison, it is necessary to repeat the experiment at least five times with the same sequence.

5.2.1.2. Stage 1 - Dynamic object segmentation

We experiment our stage 1 implementation [3.2.2.2] for dynamic object removal in a manner identical to that described above. Five trails were taken, and the average of RMSE values are shown in the table 5.2.

Sequence Number	Stage 1 - Trajectory error in meter					
	T1	T2	T3	T4	T5	Average
04	1.37	1.33	0.62	0.62	1.105	1.009
05	4.61	6.75	4.77	5.56	4.61	5.26
07	2.03	1.82	4.11	1.93	1.72	2.32

Table 5.2: Absolute trajectory error - Stage 1 experiments

The obtained results of stage 1 is compared with baseline results obtained from ORB-SLAM and DynaSLAM. Overall, our results are in line with the dynaslam except on sequence 05. From the graph, we could infer that our tracking error is less than base ORB-SLAM on sequence 04, a dynamic sequence, which complements the proposed methodology for the removal of dynamic objects. On the other hand, sequences 05

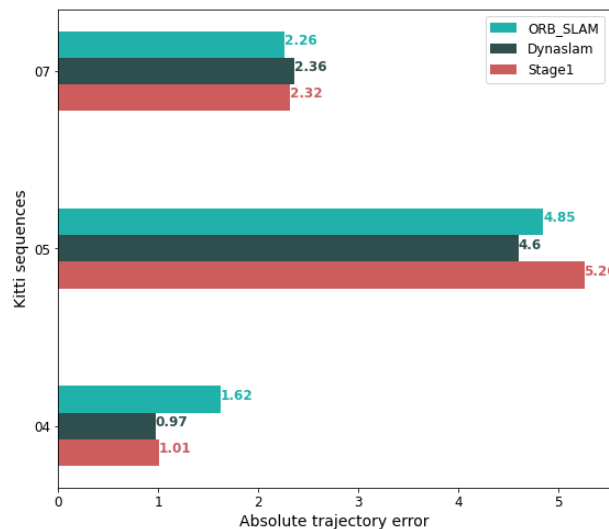


Figure 5.15: Tracking error comparison - Baseline vs Stage 1

and 07 do not provide satisfactory results. This is because the chosen sequence has a lower level of dynamic interaction since most vehicles are stationary. Using our stage 1 module would remove all feature points associated with the potential dynamic class without differentiating moving and non-moving objects separately. Elimination of

static(stationary cars) features cause the SLAM to use low textured features present at the far distance, impacting overall tracking results.

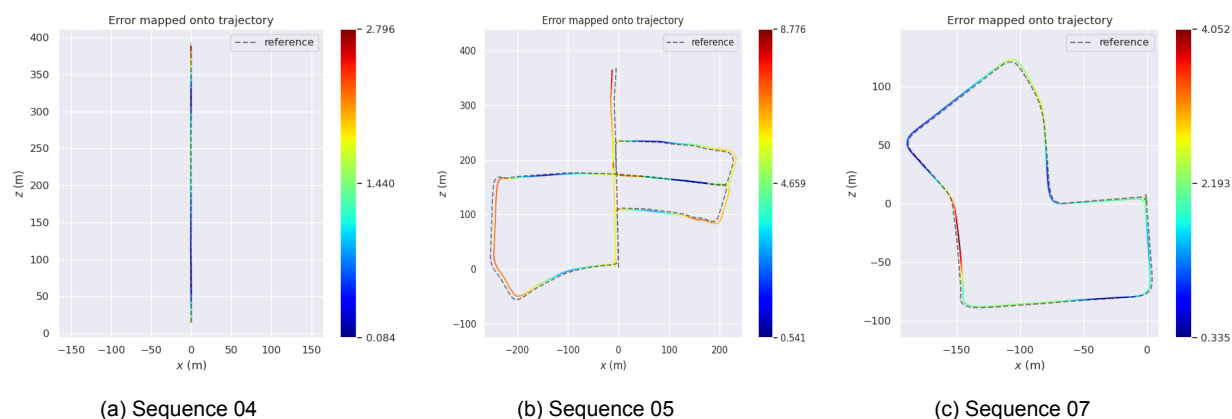


Figure 5.16: Tracking result - Stage 1

Discussion: From equation 3.1, we know that estimated poses and the landmarks of ORB-SLAM2 depend on features' quality and quantity. Many seemingly dynamic classes may exist in the environment without any movement in certain situations, i.e., they all are static. Since our segmentation model does not distinguish dynamic classes into moving and non-moving objects, all the features of dynamic classes are removed during SLAM tracking. To investigate this issue further, we used KITTI sequences and analyzed the existence of non-moving dynamic objects. For instance, in KITTI odometry sequence 05, it is observed that 95% of cars are static. It is possible to infer from the figure 5.17 that the majority of the vehicles on the left side of the image are motionless. By excluding features associated with static vehicles(non-moving dynamic class), the features available for tracking become skewed toward the right side of the image. This may impair tracking quality and also make the map sparser.



Figure 5.17: Limitation of Stage 1 - Kitti sequence 05

5.2.1.3. Stage 2 - Moving object segmentation

The limitation of the stage 1 methodology put forward the need for moving object segmentation (stage 2) [3.2.3.2] in the ORB-SLAM2 method. To prevent bias due to dynamic object removal in low traffic situations, we must correctly find moving objects from non-moving objects.

Even though the moving segmentation model results have some false positives in

Sequence Number	Stage 2 - Trajectory error in meter					
	T1	T2	T3	T4	T5	Average
04	1.53	1.14	0.96	1.26	1.35	1.25
05	4.85	4.37	4.85	4.97	4.93	4.79
07	2.61	1.99	2.03	1.62	2.25	1.97

Table 5.3: Absolute trajectory error - Stage 2 experiments

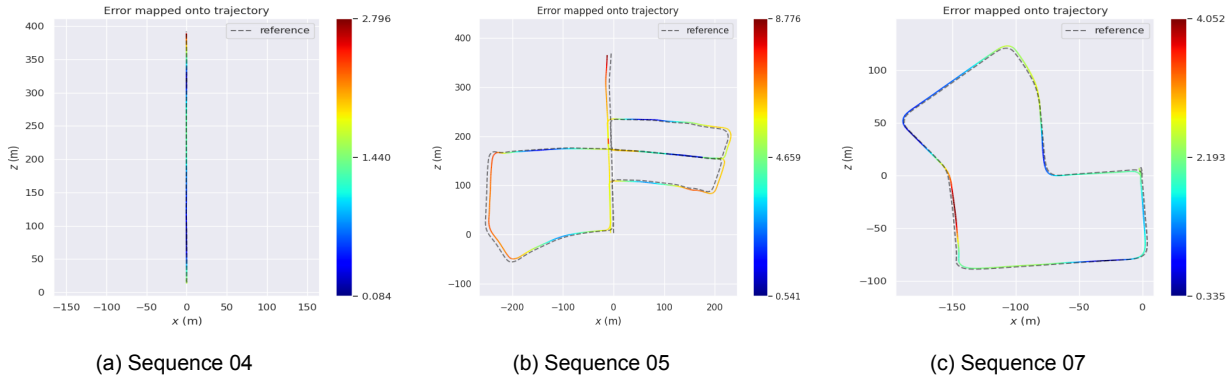


Figure 5.18: Tracking result - Stage 2

identifying the moving objects, the odometry results demonstrate a substantial increase in the performance of sequences 05 and 07 compared to stage 1. All the cars in sequence 04 are moving; hence in stage 1, removing the features of vehicles improved the tracking accuracy but due to misclassification of moving object segmentation tracking error gets higher in stage 2 implementation on sequence 04.

Discussion: The results demonstrate that the implementation of moving object segmentation could handle dynamic interaction in diverse traffic conditions without impairing SLAM's overall performance. The limitation of stage 2 implementation comes directly from the output of segmentation achieved from the two-stream network. As mentioned in moving object segmentation 5.1.2, ego-motion compensation is required for the proper estimation of flow image (input to the model).

Finally, the individual trajectory obtained for the sequences on each developmental stages are compared. From the figure 5.19, all our implementation were able to follow the reference trajectory in all three dimensions with minimal variation. Though the deviation along lateral direction(x) in sequence 04 appears to be prominent in the graph, the magnitude of the variation is negligible. The violin plot 5.20 emphasising the variation of trajectory errors seen at every timestamp reveals that stage 2 implementation on sequences 05 and 07 achieves lower error. The mean density on 05_stage2 and 07_stage2 concentrated on low ATE values. Our work authenticates a new research direction for improving the performance using loosely linked coupled approaches rather than primarily depending on tightly coupled approaches, which will aid in attaining real-time performance in the near future.

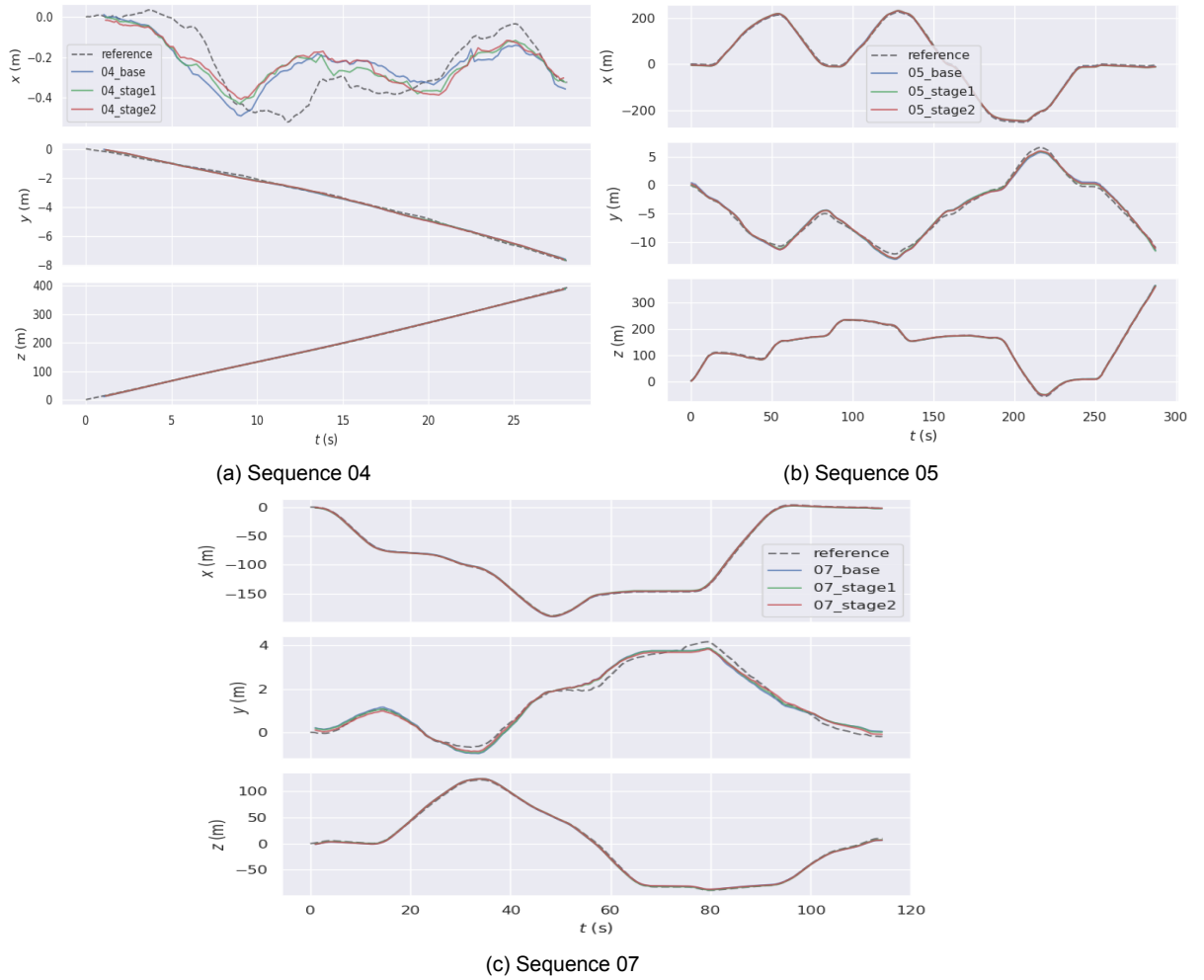


Figure 5.19: Individual trajectory comparison along x,y,z direction

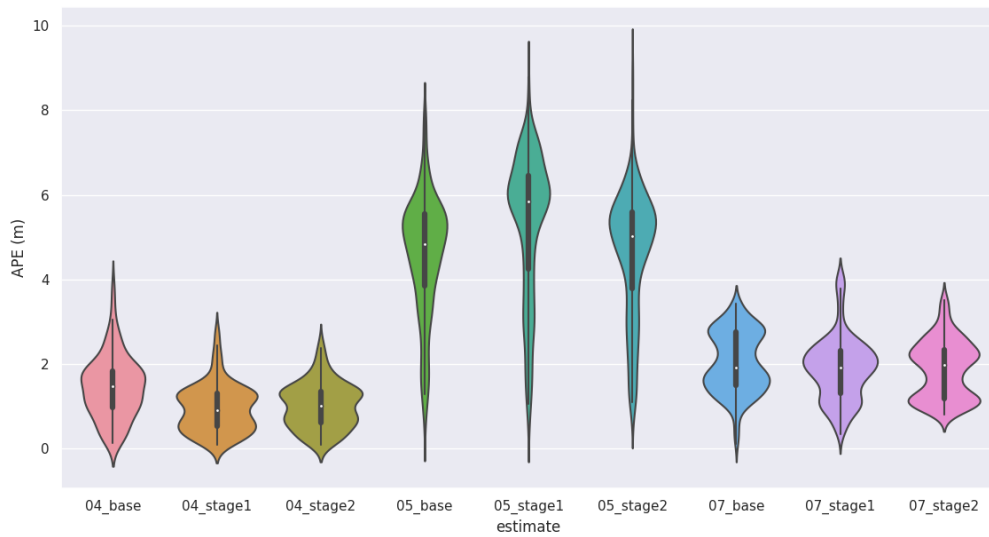
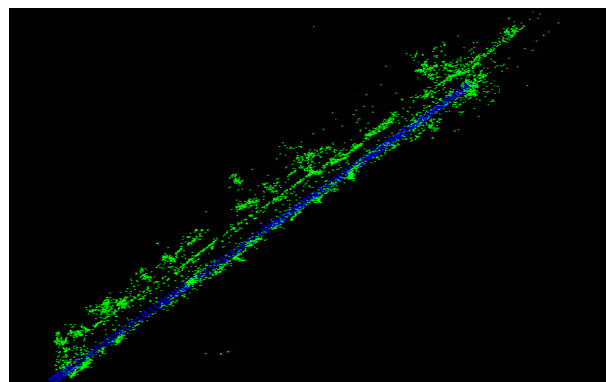


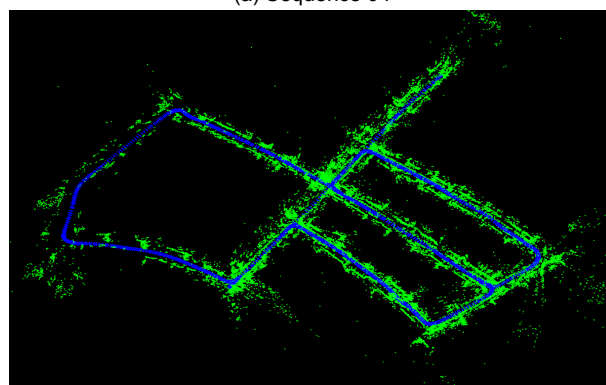
Figure 5.20: ATE plot - Base vs Stage1 vs Stage2

5.2.2. Mapping

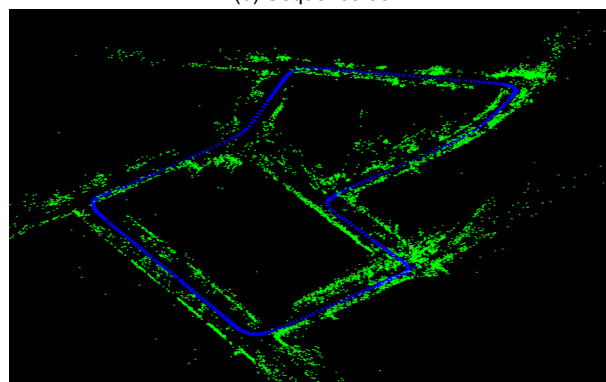
The fundamental building block of SLAM is to determine the static points in the surrounding environment. This is achieved through the normal implementation of the ORB-SLAM2 algorithm. The visualisation of the static maps obtained for KITTI sequence 04,05 and 07 on base ORB-SLAM2 is shown in figure 5.21. Green colour points are the static maps, whereas blue frames represent the keyframe poses of ORB-SLAM2. The mapping section focuses on improving the static maps by incor-



(a) Sequence 04



(b) Sequence 05



(c) Sequence 07

Figure 5.21: Base static map of ORB-SLAM2

porating the semantics learnt in stage 1 and stage 2, thereby building dynamic and instance semantic maps. Later visualisation result of dynamic density estimation and managing the maps for efficient storage are discussed.

5.2.2.1. Dynamic Semantic mapping

The dynamic subset removed during the tracking in stage 1 and stage 2 are used in the mapping to generate a dynamically meaningful representation of the surroundings. Stage 1 splits the extracted features into static and potentially dynamic subsets. Based on these subsets, a dynamic map is created on KITTI odometry sequences where stationary points are represented in green and potentially dynamic points in red as shown in the figure 5.22. Along with the feature data association, the dynamic information of each map point also needs to be correlated with respective keyframes. These implementations allow the poses and map to be optimised through bundle adjustment. Though we build maps for all the selected sequences, only KITTI sequence 07 is used to discuss the result in the rest of this section. The dynamic map built based on the output of stage 2 has three subsets. The effect of our stage 2 implementation on KITTI odometry sequence 07 shown in the figure 5.23 has static points, non-moving dynamic points and moving dynamic points.

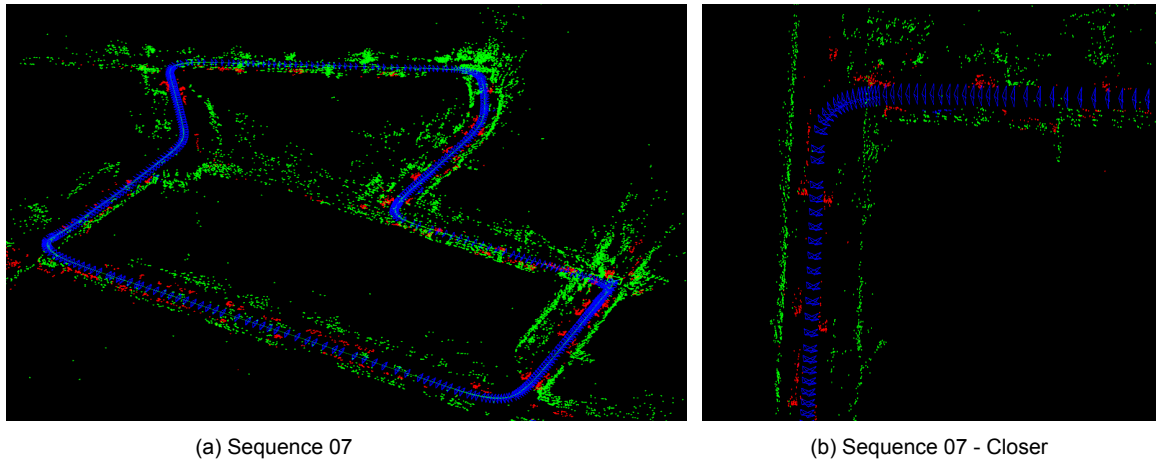


Figure 5.22: Dynamic semantic map - Stage 1. Red - Potentially dynamic points, Green - static points

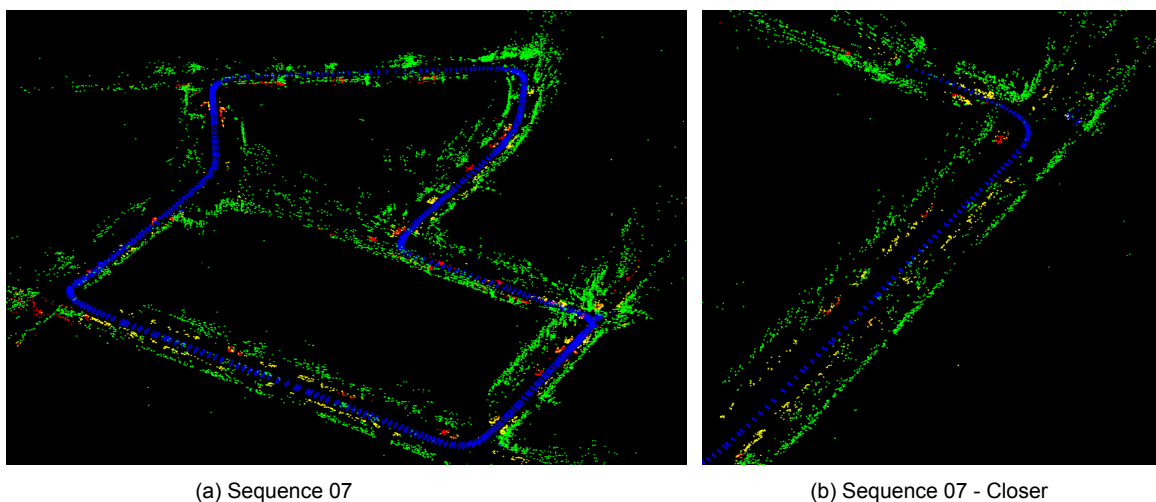


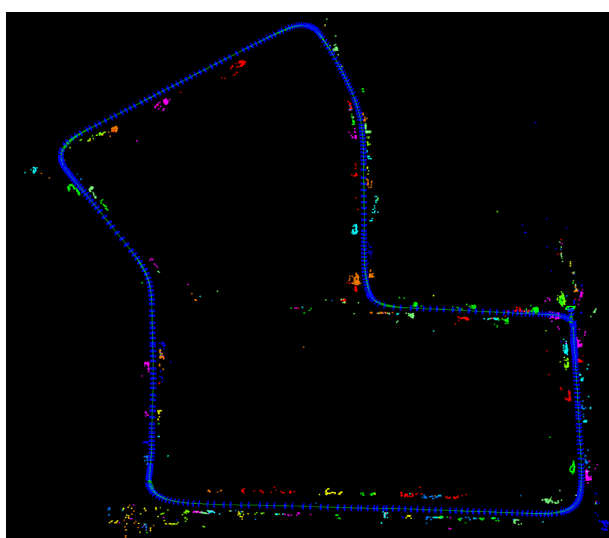
Figure 5.23: Dynamic semantic map - Stage 2. Red - moving dynamic points, Yellow - Non-moving points, Green - static points

The map obtained with stage 2 output is not accurate due to the limitation of moving object segmentation. The false-positive segmentation degrades the semantic quality when distinguishing moving and non-moving entities. Nevertheless, the maps are still accurate in finding the objects that belong to the car.

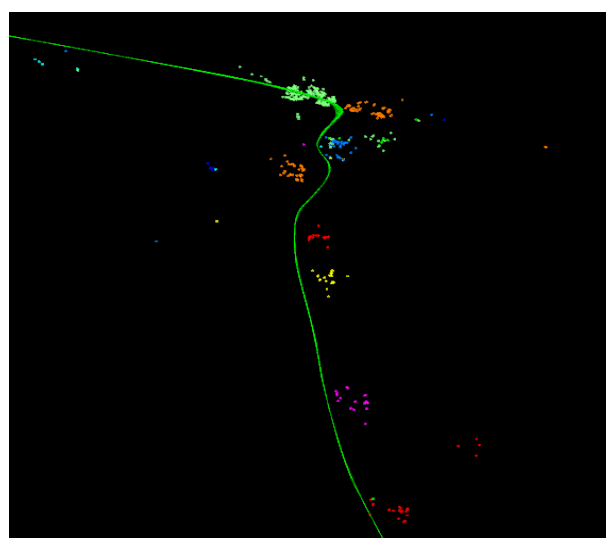
According to the research [15], ORB-SLAM2 allows lifelong mapping because of the ability to accurately relocalize in the learnt map even when the viewpoint shifts significantly. Our dynamic semantic mapping could improve the robustness in scene perception and make relocalization more successful by considering only static points eliminating dynamic entities.

5.2.2.2. Instance semantic mapping

While the dynamic semantic mapping distinguishes dynamic classes from static points, it is often necessary to analyze individual objects in the environment. Instance semantic maps helps to achieve desirable results on identifying individual instances in the 3D space. Multi-object tracking identification is used to relate map points that belong to the same object. The results shown in the figure 5.24 demonstrate that our implementation could correctly find the presence of an individual car in the surroundings. Distinct colours is used to represent the multiple instances of cars. The figure 5.24a represents the overall visualization of instance semantic map on KITTI sequence 07, excluding the static points. While the figure 5.24b is the expanded representation of instances of cars identified in KITTI sequence 05.



(a) KITTI sequence 07



(b) KITTI sequence 05 - Closer look

Figure 5.24: Instance semantic maps - KITTI sequences

5.2.2.3. Dynamic density estimation

The benefit of the dynamic semantic map is exploited to represent the density information to motion planning robots. This module is implemented to show the usefulness of three-dimensional semantic reasoning in a real-time scenario. The dynamic density

estimation done based on spherical bounds are represented as a circle is shown in the figure 5.25, where yellow and red denotes low and high dynamic density, respectively. During experimentation, one important observation is that the density estimations of previously mapped areas are more accurate than those of newly discovered areas. This is very apparent since the newly created region may contain fewer map points to study the dynamic interaction of the local map.

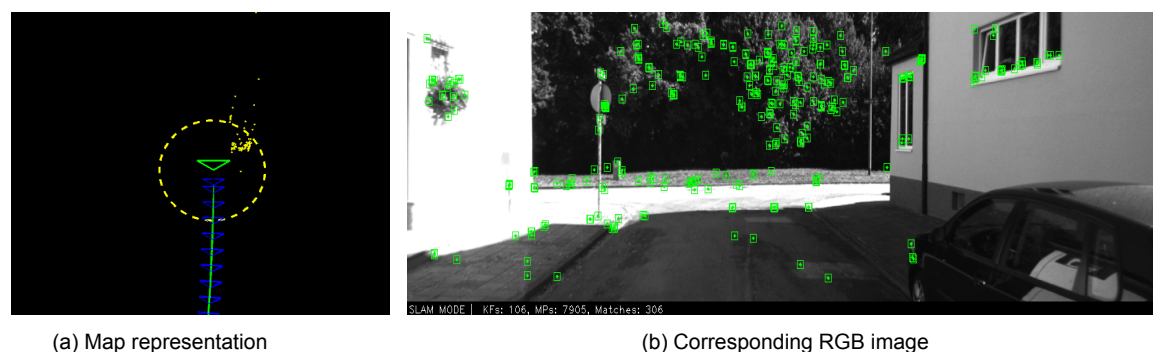


Figure 5.25: Low dynamic interaction example



Figure 5.26: Dynamic density estimation

5.2.2.4. Map management

By default, ORB-SLAM2 does not support map storage and it only allow us to visualize the map points during the SLAM operation. This section discusses our implementation to save maps in point cloud libraries(PCL) [85] and octomaps [86] by untangling the details of both libraries based on the storage efficiency.

PCL is an open-source library that is designed to handle point clouds efficiently. PCL supports different data structures for storing point clouds, among which our implementation make use of PointXYZRGB to save the final map. In the data structure XYZRGB, each map point requires three-dimensional coordinate values for x, y, and z, as well as three color intensity information for RGB. x, y, and z values are directly obtained from the coordinates of the global map generated by ORB-SLAM2. While RGB information helps to encode the dynamic semantic information of each points. For instance, the output of stage 1 contains two different categories which are colour

coded as $\text{green}(x,y,z,0,255,0)$ and $\text{red}(x,y,z,255,0,0)$ in PCL data structure. All the map visualisation shown in previous chapter are based on the point cloud representation.

OctoMap uses voxel-based representation (based on leaf resolution) to store map efficiently. Octree provides a novel solution to represent maps in cubic volumes with tree-based architecture designed to update the map in a probabilistic fashion. The map can be dynamically expanded and also be saved in multi-resolution within the given cubic space. The representation provides a proper distinction between free, occupied, and unknown areas. This kind of representation reduces the computational complexity when retrieving certain points in the map. Changing the leaf size would alter the input resolution and occupancy probability in 3D space. The figure 5.27 shows the octomap representation for different leaf size on KITTI odometry sequence 07.

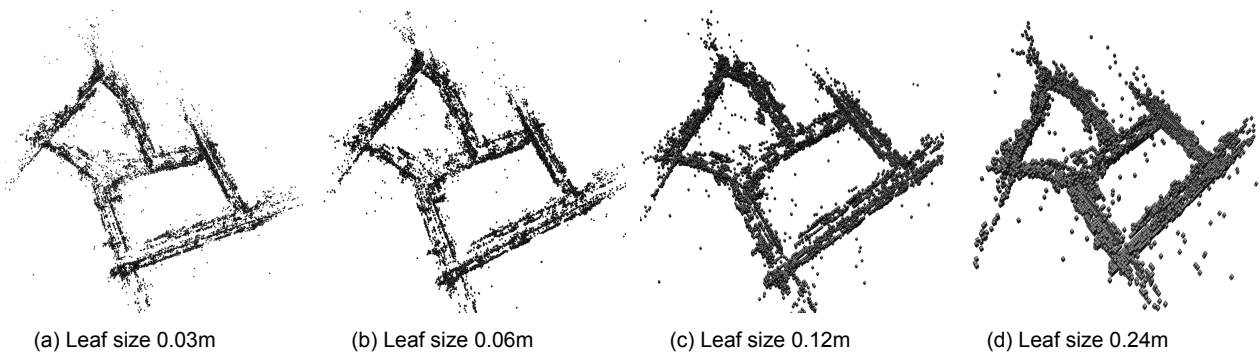


Figure 5.27: Octomap with different resolution

PCL 1.12.0 library is accommodated in our pipeline to store the maps acquired from the ORB-SLAM2. The PCL library has a direct extension to save the octomap without any hassle. The system class function of ORB-SLAM2 is modified to loop through the global map points after global bundle adjustment, ensuring that the final optimized map points are stored. The map management does not consider the estimated poses when the map is stored. The comparison of memory needed to save the map as point cloud library and octomap with high resolution (leaf size 0.03m) is shown in the table 5.4.

Sequence No.	Size of PCL (in KB)	Size of Octomap (in KB)	% Reduction
04	328	13.1	96
05	2619	82.5	96.8
07	1275	67.58	94.69

Table 5.4: Map memory management

The table shows the benefit of octomap based on the percentage of reduction in the memory requirement. Reduced memory footprint enables efficient implementation for large-scale mapping in embedded devices such as NVIDIA Xavier.

5.3. Real-world validation

We conducted real-world experiments to demonstrate the use of semantic mapping in a diverse scenario. Instead of classifying dynamic objects in the scene, the implementation is extended to identify the goal state in three-dimensional space. We gathered real-time data of our robot dynamics lab with the husky robot using zed2 cameras as shown in the figure 5.28.

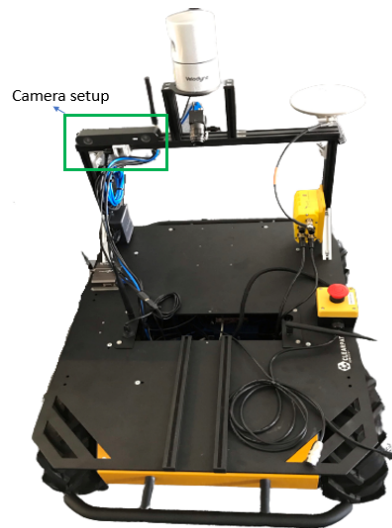
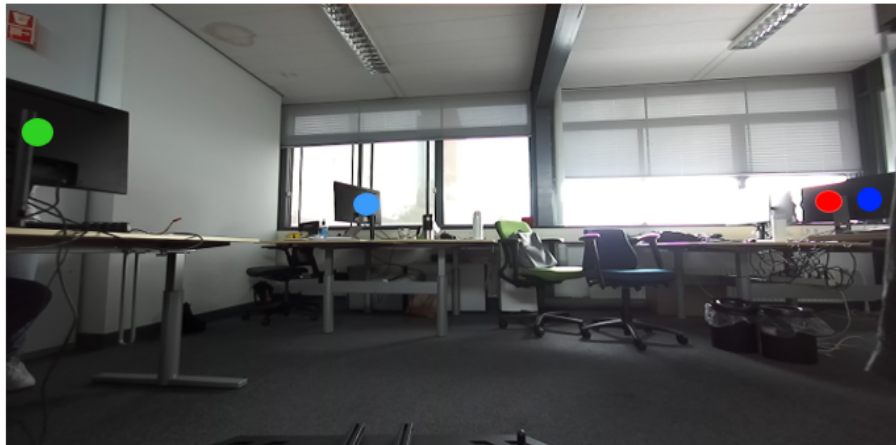


Figure 5.28: Robot setup used for real world experiment

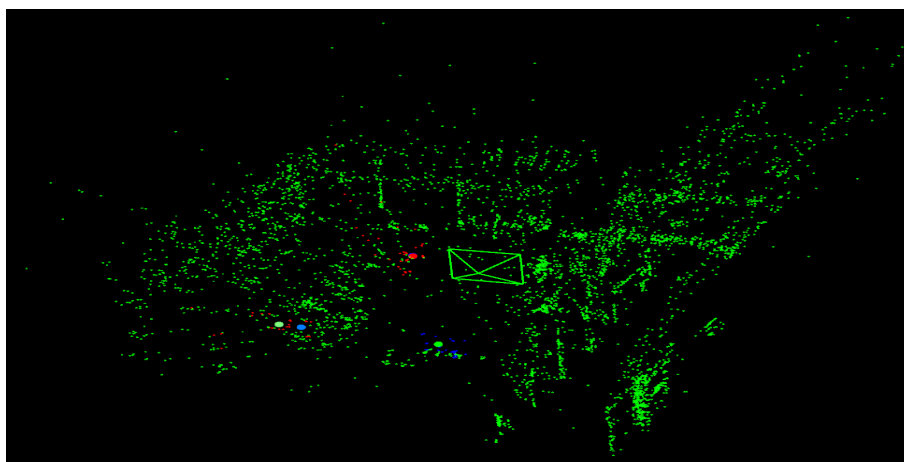
For the demonstration, all the monitors present in the environment are set as our object of interest. Similar to the semantic association of car discussed in the earlier section, a map is built to identify the monitor in 3D space. Along with determining the semantic points associated with the monitor, we estimate the centroid of each monitor individually to locate the target point in the three-dimensional space. However, outliers from the mapping process of ORB-SLAM2 would estimate the centroid with error. This prevents the estimation of centroid directly from the obtained point clouds. To improve the centroid calculation, the following conditions are employed,

- Only the subset of map points that belong to the same instance-id is selected for calculation. The output of instance semantic mapping is used for isolating the subset.
- Subset representing individual object must at least have 20 map points.
- Distance between the map points in the subset and latest reference keyframe must be within a certain threshold.

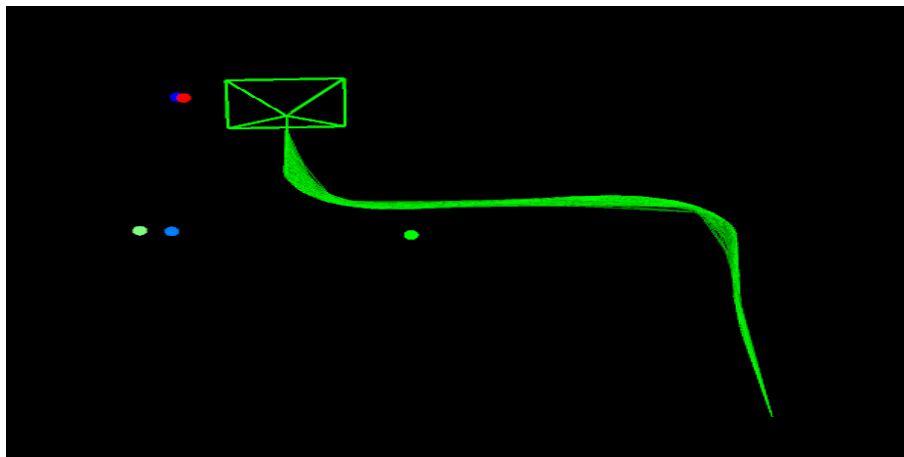
The results attained on real world data reveals that our implementation could be easily generalised to various applications, such as identifying a target location or avoiding specific barriers in dynamic object removal.



(a) Monitors in the lab



(b) Mapping result



(c) Tracking result with centroid estimation

Figure 5.29: Real world experiment result - Tracking and mapping

6

Conclusion

This project thrives on the objective to build robust interactive maps for navigation in a dynamic environment. The methodology discussed in the thesis is systematically developed to achieve the final goal by answering the research questions.

1. What is a popular approach to overcome the challenges of dynamic objects in SLAM?

From the literature study, a learning-based approach was identified as a suitable methodology for removing the features that correspond to dynamic objects. To that extend, we investigated the benefits of single stage instance segmentation architecture SOLOV2 with other popular architectures(Mask-RCNN) adapted in SLAM. Experimental results show that SOLOV2 is faster and gain 10.27% higher accuracy than Mask-RCNN.

2. How to improve the quality of tracking and mapping with dynamic interaction?

The evaluation of dynamic object removal using SOLOV2 on the KITTI sequence reveals that our proposed method yields good odometry results in the presence of high dynamic interaction. However, in low traffic scenarios i.e., when most of the cars are static in the given sequence, tracking results have deteriorated. This insists on classifying the potential dynamic objects into moving and non-moving entities.

We developed moving object segmentation architecture based on MODNet paper which uses monocular image and optical flow to find moving and non-moving cars. Though the results obtained from our model have false positives, the methodology enhances the tracking results considerably in a low dynamic situation. Improving the accuracy of the moving object segmentation module would open a new path for other researchers who works on real-time isolation of moving entities.

3. How to build a semantically meaningful map?

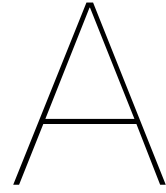
In contrast to other studies which removes the dynamic features entirely from the SLAM processing, we eliminate dynamic features during the tracking stage of SLAM but retain them during the mapping stage. This enables us to develop semantically

meaningful dynamic maps, which facilitates understanding of traffic interactions in three-dimensional space.

In order to enhance semantic reasoning further, multi-object tracking is combined with instance segmentation to produce instance semantic maps. The overall pipeline for multi-object tracking was built to process the segmentation output independently and generate object correspondence between the detected masks of two consecutive images. The qualitative results of object tracking on images and three-dimensional instance semantic map demonstrates the potency of the proposed approach. The real world experiments on the robot manifest the effectiveness of our interactive mapping and highlight the generalisation of our method on various application scenarios. Finally, the map management module examines various approaches to store and efficiently handle maps.

6.1. Recommendations

- In our study, the monocular camera is used as the primary sensor modality to experiment the dynamic object removal and semantic mapping. The behaviour of our proposed methodology needs to be studied on stereo and RGB-D cameras.
- Experiments of stage 2 implementation on SLAM infers that moving object segmentation improves tracking marginally. Poor segmentation results from moving segmentation restrict us to validate the actual benefits of moving and non-moving classification. The inaccuracy of moving object segmentation is mainly due to the limitation of ego-motion compensation during optical flow image estimation. Further research needs to be conducted to determine appropriate ego-motion to compensate for the influence of moving cameras.
- The observational study on the newly created dataset for moving object segmentation using KITTI sequences reveals that moving cars are concentrated on the centre of the image. Also, the current dataset is annotated to capture only moving car, not extended to other classes. Henceforward, the dataset needs to be created with diverse classes with a proper spread of moving objects across various positions in the image.
- Multi-object tracking tracks the segmentation instances just between two consecutive frames. In case of occlusion or variation in the lighting, particular objects are missed from the monitoring. If the same object is identified in the next frame, it will switch the identification number to a new id. In future studies, the search space of the tracking sublet needs to be increased from two to at least ten frames to reduce the tracking id switch.
- Currently, the deep learning models used in tracking, stage 1 and stage 2, are processed as separate modules. Joint training and evaluation would speed up the inference time and improve the final accuracy.



Appendix

This chapter explains the technical concepts and empirical theory of SLAM and deep learning models that are required to understand the overall implementation done in this project to achieve the research goal. The structure is divided into two sections, the first part discusses about SLAM and the second one covers technical details of neural networks.

A.1. SLAM Background

A.1.1. Feature matching

Image is the collection of light intensity values represented in matrix form. The change in intensity values provides distinctly useful information that is interpreted as visual features. These features are the key for constructing data correspondence between sequential frames. Identifying a single pixel as a feature might not be robust to track the correspondence in the real world as the intensity value changes due to illumination. Hence, key points are introduced to find distinct locations and descriptors to understand the key points' local structure. Key points are generally the corners where the gradient change is sharp and distinguishable from other places in the image. In the real world, these features are not sufficient to identify the appropriate representations. A bag of visual words with handcrafted features that make the keypoint detection robust in the practical condition. SIFT [87], SURF [88] and ORB [89] are some of the commonly used feature matching techniques that deliver potent results. Extracted features should be robust to change in transformations such as illumination, rotation and scale. Each method have their own advantage and limitations; a detailed review of all the keypoint estimation is not the area of focus. Instead, detailing the ORB feature would be sufficient to understand the pipeline used in the project.

ORB(Oriented Fast and Robust BRIEF) uses FAST [90] and Harris corner measures to identify the dominant key points. Features extracted from the FAST do not embrace orientation and scale. ORB solves the scaling problem by employing the image pyramid to extract FAST features at each pyramid level. The direction of the vector between corners and the intensity weighted centroid orients the Fast features. ORB uses rotation aware BRIEF(Binary Robust Independent Elementary Feature) as their

descriptor. This binary descriptor enables faster processing with additional rotation information. ORB features extracted between two images is used to find the feature correspondences in this project.

A.1.2. Coordinate transformations

Let the coordinate system used in the SLAM represented as world (w), camera (k), image (i) and sensor (s) coordinates. The transformation of 3D point in world coordinate system to a 2D point in sensor coordinates is computed based on the extrinsic and intrinsic parameters,

$$\begin{bmatrix} x^s \\ y^s \\ 1 \end{bmatrix} = E K \begin{bmatrix} x^w \\ y^w \\ z^w \\ 1 \end{bmatrix} \quad (\text{A.1})$$

where:

$$\begin{aligned} E = T_w^k &= \text{extrinsic matrix (world} \rightarrow \text{camera coordinate)} \\ K = T_c^s P_k^c &= \text{intrinsic matrix (camera} \rightarrow \text{image} \rightarrow \text{sensor coordinate)} \\ T &= \text{transformation matrix} \end{aligned}$$

Extrinsic parameters involve a 3D transformation of world to camera coordinates which has 6 degrees of freedom, three for rotation and three for translation. Let point in the world coordinate be ${}^w X_p (x^w, y^w, z^w)$ and camera centre be ${}^k X_o (x^k, y^k, z^k)$, then the point in camera coordinates is related to world coordinate based on the transformation as,

$$\begin{aligned} {}^k X_p &= T_w^k {}^w X_p \\ {}^k X_p &= R({}^w X_p - {}^k X_o) \end{aligned} \quad (\text{A.2})$$

Here R denotes rotational component of transformation. Usually homogeneous coordinates are convenient in expressing the rotation and translation as a simplified matrix which makes the computation easier. In homogeneous coordinates equation A.2 could be expressed as,

$${}^k X_p^h = R[I \mid - {}^k X_o^h] {}^w X_p^h \quad (\text{A.3})$$

Intrinsic parameters involve 3D to 2D transformation from camera to image coordinates and 2D to 2D transformation from camera to sensor coordinates. 3D-2D transformation is also known as perspective projection. Transformation to sensor coordinates is usually taken care by adding the deviation in the intrinsic parameters along x and y axis. In an ideal scenario, transformation by the perspective projection matrix without deviation is given by,

$$\begin{aligned} {}^c X_p^h &= P_k^c {}^k X_p^h \\ {}^c X_p^h &= \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} {}^k X_p^h \end{aligned} \quad (\text{A.4})$$

A.1.3. Epipolar constraint

Consider a 3D point P in the world observed as the features point p_1 and p_2 in the camera coordinate of two individual frames respectively, as shown in the figure A.1.

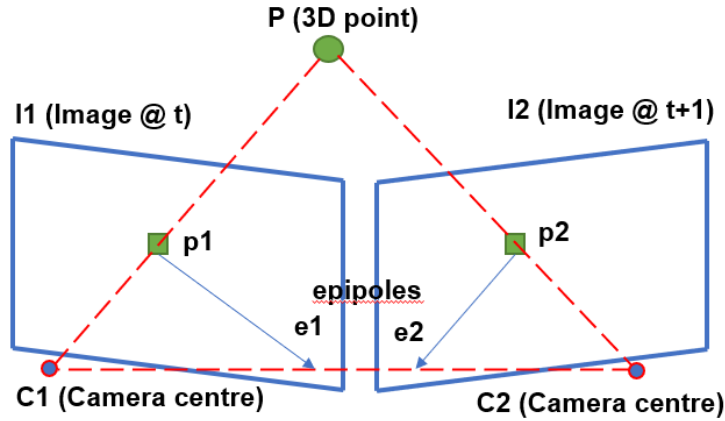


Figure A.1: Epipolar geometry

The plane formed between the two camera centres and point P is called the epipolar plane. From the figure A.1, it could be observed that point P could lie at any location on the projected ray from the feature p_1 . The exact location of point P could be identified by the intersection of the ray projected from feature p_2 and p_1 . Hence, the accuracy of the estimated point location depends on the quality of feature matching. Relation between feature points in two images could be established using camera motion model represented as the transformation matrix (rotation R and translation t) [91].

$$p_1 = R.p_2 + t \quad (\text{A.5})$$

Cross product with t , then multiplying p_1 on both sides we modify the equation into,

$$\begin{aligned} p_1(t \times p_1) &= p_1(t \times R.p_2) \\ p_1^T [t_x] R p_2 &= 0 \end{aligned} \quad (\text{A.6})$$

Equation A.6 expresses the relationship between the point in the camera coordinate. The translation and rotation matrix could be represented together as the essential matrix E ,

$$p_1^T E p_2 = 0 \quad (\text{A.7})$$

Based on the relationship between essential and fundamental matrix F we can extend the epipolar constraint between two points u_1 and u_2 in the sensor/image coordinates. Fundamental matrix correlates the same points observed from an scene in two different frames.

$$\begin{aligned} (u_1^T K^{-T}) E (K^{-1} u_2) &= 0 \\ u_1^T F u_2 &= 0 \end{aligned} \quad (\text{A.8})$$

Equations A.7 and A.8 are interchangeably used as epipolar constraints where epipolar lines is given by $l = F u_2 = F^T u_1$. Fundamental matrix and essential matrix found using eight-point [92] or five-point algorithms [93]. Under pure rotation, it is

hard to estimate the Essential and Fundamental matrix. In such cases, homography matrix helps to relate a point in an image with another image. Homography is a projective transformation of planar objects represented in a three-dimensional matrix. The idea of fundamental matrix, epi-polar geometry, homography would be used in various components of our SLAM module.

A.1.4. Triangulation

Triangulation estimates the depth in monocular SLAM to an arbitrary scale for estimating 3D map points. It uses epi-polar constraint to estimate 3D location from the intersection of rays originated from feature points. Error in feature extraction could cause the rays not to intersect. Re-projection error is used to ensure the quality of 3D points by back-projecting rays onto the image co-ordinate with 2D feature location. The obtained map points are refined along with camera poses later through bundle adjustment methods. Details of re-projection error and bundle adjustment are discussed in next section. The main disadvantage of triangulation is that it does not work well with pure rotation.

A.1.5. Bundle adjustment

Bundle Adjustment (BA) [94] is an optimisation procedure to find the optimal transformation that corrects the accumulated drift during visual odometry. Bundle adjustment is the minimisation problem that corrects the estimated poses and map points based on the re-projection error. If the 3D point P_i projected on the camera j (say p_{ij}) is compared with 2D image coordinate measured from the camera \hat{p}_{ij} , then reprojection error is given as,

$$\min \sum_n \sum_n^{i=1, j=1} (p_{ij} - \hat{p}_{ij}) \quad (\text{A.9})$$

The analytical method cannot be used to solve the equation due to high degree of non-linearity caused by the geometrical constraints associated between poses and corresponding map points. Hence, iterated methods [95] such as Gauss-Newton or Levenberg-Marquardt algorithm are used to estimate corrected R , t and P_i . Levenberg-Marquardt algorithm [96] bounds the minimisation problem by a damping factor which chooses the optimisation trend between least square and steepest descent, achieving faster convergence. Sparsity observed in SLAM ensures the optimisation could run in real-time by matrix decomposition. SLAM researchers use G2o (General graphic optimisation) [97] library to solve such non-linear optimisation.

A.1.6. Graph based SLAM

The graph-based system enables usage of SLAM in large-scale scenarios. Graph-based architecture is composed of vertices where each vertex (node) in the graph corresponds to the poses or landmarks. All nodes are connected by edges which represent the spatial constraints between the poses/landmarks obtained from the camera measurements. Usually, the keyframes that observe similar feature points are connected with each other. Graph-based SLAM usually has a front-end for finding the

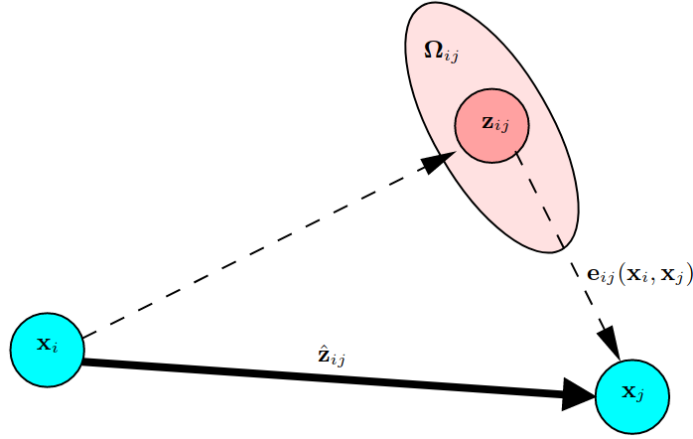


Figure A.2: Graph based SLAM [98]

relative transformation of poses to construct graph and back-end bundle adjustment to correct the drift accumulated due to reprojection error. Let x_i be a node that observes node x_j by estimating the relative transformation \hat{z}_{ij} between these two nodes. Actual measurement z_{ij} and corresponding information matrix Ω (inverse covariance matrix) identifies the expected location of these poses in the graph as shown in the figure A.2. Information matrix adds weights to each node during the final optimisation, where the optimisation involves minimising the loglikelihood. Then the minimisation problem could be expressed as follows which is then solved using one of the popular iterative methods discussed in previous section,

$$\hat{X} = \underset{X}{\operatorname{argmin}} \sum_{ij} e_{ij}^T \Omega_{ij} e_{ij} \quad (\text{A.10})$$

$$e_{ij}(x_i, x_j) = z_{ij} - \hat{z}_{ij}$$

A.1.7. Optical flow

Optical flow is the estimation of relative motion of pixel intensity values captured between two camera instances. For the static camera without dynamic objects in the scene, optical flow is calculated as zero. Optical flow algorithms have two fundamental assumptions [99],

- Temporal persistence - motion of any given point between two consecutive image frames should be relatively small.
- Brightness constancy - the intensity value of a pixel remains constant between two subsequent images.

If a point represented as x and y in an image which moves with velocity v from time t to $t + dt$, then an image could be represented based on the above assumptions as,

$$I(x + dx, y + dy, t + dt) = I(x, y, t) \quad (\text{A.11})$$

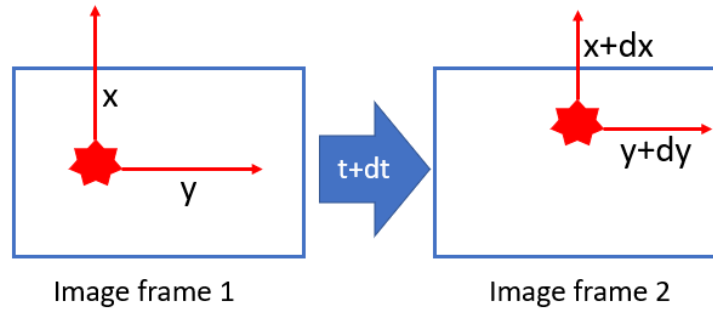


Figure A.3: Optical flow explanation

Using Taylor expansion and associating the first order derivatives to zero based on the temporal consistence property, the optical flow equation is given by,

$$\frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} = 0 \quad (\text{A.12})$$

where:

u, v = flow vectors along x and y direction

Equation A.12 is under-constrained, and it is hard to obtain a direct solution. Lucas-Kanade or Horn-Schunck method assumes the local and global smoothing respectively to over-constrain the equation A.12, which helps in calculating the optical flow vectors. Deep learning methods such as FlowNet [100], PWC-Net [101], ARflow [102] are used as a substitute to the conventional methods achieving the state of the art result. These deep learning models captures the temporal variation of the image as horizontal and vertical flow vectors in two channels. These vector representation are converted as RGB images as shown in the figure A.4 for easy interpretation.

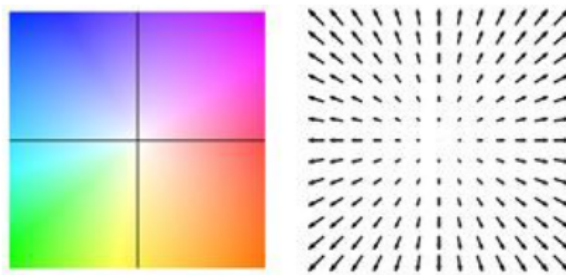


Figure A.4: Optical flow - Color representation [61]

A.2. Deep learning Background

Artificial intelligence brings a new dimension of possibilities to make the computer imitate human behaviour. Machine learning is a popular subset of artificial intelligence where the computer learns to differentiate among the given features to make suitable decisions. Deep learning is a more specific area of machine learning where the model

trains to learn the features directly for the given task. Inherited representation learning in deep learning models overcomes the main problem of identifying the features separately. Training the model, maps the input (x) to the output (Y) which is generalised as

$$Y = f(x, \theta) \quad (\text{A.13})$$

where:

θ = parameters learnt by the model

The deep learning method is categorized into supervised, semi-supervised, or unsupervised based on the availability of ground truth information in the training data [103]. Dataset of supervised learning contains target label for individual input based on the final task. The model trained on supervised dataset learns the conditional probability $p(Y|x)$ based on the error between prediction and output labels. Unsupervised learning trains the model in the absence of specific output labels and therefore it attempts to determine the overall structure of the dataset. For a given input data, unsupervised techniques determine joint distribution $p(x, Y)$. Unsupervised learning is typically employed in clustering, dimensionality reduction, and generative networks. Semi-supervised learning is a hybrid of supervised and unsupervised learning best suited for situations with a small number of labelled and many unlabeled data. In our study, we employ supervised learning approaches to train artificial neural networks to obtain the required outcomes.

A.2.1. Neural networks

Let f represent map function that ideally correlates the input to output. To effectively learn the representation of the map function we define the model, cost function and optimization methods. Neural network model is usually the stack of multiple layers where the overall function f is made up by the combination of features learn from the intermediate layers represented as

$$f = f_1(f_2(f_3, \dots(x, \theta))) \quad (\text{A.14})$$

where:

f_1 = first layer

f_2 = second layer

f_3 = third layer and goes on

Before diving deeper into the different model architecture, the functionality of the primary component that made up the whole architecture is discussed briefly.

A.2.1.1. Neuron

Neurons are the basic building block of neural networks that store the parameter value as weight and bias during training [104]. Many neurons act in parallel with each other and final output is calculated as weighted sum of inputs with a bias term. The equation below depicts a linear function in which non-linearity is added by activation function.

$$Z = \sum_i W_i^T \cdot x_i + b \quad (\text{A.15})$$

where:

w = weight term

b = bias term

$i = \{1, 2, \dots, n\}$

$z = \{-\infty, +\infty\}$

A.2.1.2. Activation function

The activation function attempts to behave similarly to neurons in the human brain, allowing output to pass only when it reaches a specific threshold. It also attempts to limit the output to a particular range, preventing underflow during calculation. The activation function might be linear or non-linear. Only non-linear activation functions are discussed below.

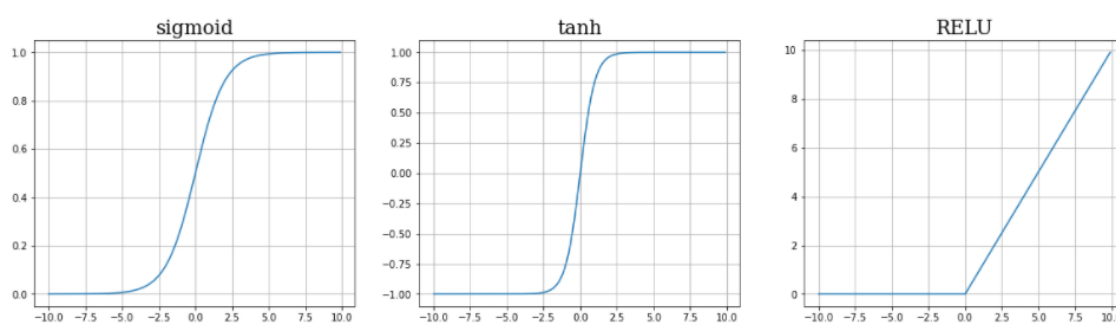


Figure A.5: Various types of activation function

Sigmoid function squeezes the output between 0 to 1. Even a small change in the input value causes significant change in the output which makes sigmoid ideally suitable for classification task [105].

$$\sigma(z) = \frac{1}{1 + e^{-x}} \quad (\text{A.16})$$

Unlike sigmoid function, **Tanh function** squeezes the output between -1 to 1, which makes gradient flow much steeper than sigmoid. Though sigmoid and tanh add non-linearity in the system, they squeeze output to a smaller magnitude which causes vanishing gradient problems for deeper model architecture [106].

$$\tanh(z) = \frac{2}{1 + e^{-2x}} - 1 \quad (\text{A.17})$$

Rectified Linear Unit (ReLU) [107] is a frequently used activation function that, to some extent, avoids the problem of vanishing gradient. Because it limits the output values between 0 and infinity, the ReLU activation may occasionally blow out the gradients. It is considerably more computationally efficient and sparse than sigmoid and tanh. The primary disadvantage of the ReLU is that it ignores negative output, leading some neurons to stay dormant. To address these disadvantages, leaky ReLU and parameterized ReLU may be viable options. Models implemented in our pipeline uses ReLU activation in intermediate layers.

$$\text{ReLU}(z) = \max(0, x) \quad (\text{A.18})$$

A.2.1.3. Loss function

Neural networks use loss function as the evaluation criterion to minimize the error between the predicted output and the ground truth. The scalar loss value calculated by the general loss function A.19 is utilized to update the model parameters iteratively until the global minimum is attained. The choice of loss function depends on the ultimate problem that the model must answer to complete the desired task. Individual details of loss functions used in training the model is discussed in the following chapters. General loss function is given by,

$$loss = L_f(\hat{y}, y) \quad (A.19)$$

where:

L_f = objective function
 \hat{y} = network prediction
 y = ground truth label
 $loss$ = scalar value

Back-Propagation: Model is trained by feeding the input in forward pass, comparing the outcomes based on the loss function, and updating the model parameters. Back-propagation is a technique for updating network parameters(weight and bias) which computes partial derivatives based on a given objective function. The updating rule based on the gradient descent algorithm is given by,

$$\begin{aligned}
 w_{(t+1)} &= w_t - \alpha \frac{\partial L}{\partial w(t)} \\
 b_{(t+1)} &= b_t - \alpha \frac{\partial L}{\partial b(t)}
 \end{aligned} \quad (A.20)$$

where:

$\alpha(> 0)$ = learning rate

A.2.1.4. Optimiser

An optimizer is a method for adjusting the weights that aids in the faster convergence. One of the significant hyper parameters of optimisation that determine the step size of gradients flow is the learning rate. Setting a low learning rate will result in a slower convergence to an optimal solution. The higher learning rate would speed up training, but the training becomes unstable when the gradient approaches closer to zero [108].

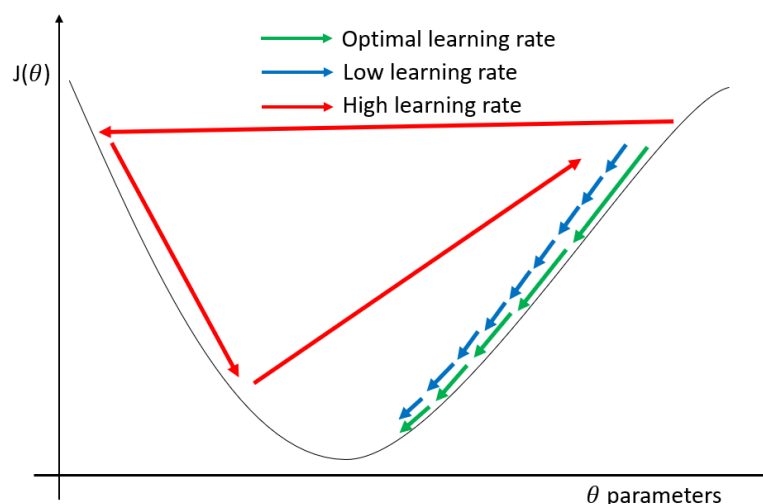


Figure A.6: Learning rate explanation

Gradient descent allows the gradient to flow in the negative direction of slope based on the error. The primary downside of this approach is that it computes the gradient for the entire dataset at once in order to optimize the model parameters, which increases the computational cost. On the other hand, stochastic gradient descent (SGD) [109] updates the model parameters based on single input data to ease the computation burden. But, the frequent updating may result in significant variation and uneven gradients update. The mini-batch gradient descent enhances SGD's capabilities by updating after each designated mini-batch.

Gradient descent algorithms have additional hyperparameters (depends on type of optimiser) to properly govern the direction of the gradient flow by decreasing variation that occur due to noise. The **momentum** term [110] is an often used to accelerate the path of gradients towards global minima, thus avoiding the gradient from getting trapped in a local minimum. The momentum term β_1 accumulates an exponentially weighted average of the past gradients in the current optimization step. The hyperparameter value β_1 of 0.9 was selected as default. **RMS-prop** [111] accelerates learning in the desired dimension by lowering variation in the other dimension using an exponentially weighted average analogous to momentum. Unlike momentum, this technique discards extreme past gradients during accumulation, allowing it to converge quicker. The **Adam** optimiser [112] is a combination of the momentum and RMS-prop algorithms that is extensively used in modern neural network implementation. It computes the gradient using first and second order derivatives and adjusts the learning rate to converge more quickly. During training, we make use of Adam optimiser for updating weights.

Based on the momentum optimiser,

$$\begin{aligned}
 v_{\partial w} &= \beta_1 v_{\partial w} + (1 - \beta_1) \partial w \\
 v_{\partial b} &= \beta_1 v_{\partial b} + (1 - \beta_1) \partial b \\
 v_{\partial w}^c &= \frac{v_{\partial w}}{1 - \beta_1^i}; v_{\partial b}^c = \frac{v_{\partial b}}{1 - \beta_1^i}
 \end{aligned} \tag{A.21}$$

Based on RMS-prop optimiser,

$$\begin{aligned} s_{\partial w} &= \beta_2 s_{\partial w} + (1 - \beta_2) \partial w^2 \\ s_{\partial b} &= \beta_2 s_{\partial b} + (1 - \beta_2) \partial b^2 \\ s_{\partial w}^c &= \frac{s_{\partial w}}{1 - \beta_1^i}; s_{\partial b}^c = \frac{s_{\partial b}}{1 - \beta_1^i} \end{aligned} \quad (\text{A.22})$$

Adam optimiser could hence be derived as,

$$\begin{aligned} w_{(t+1)} &= w_t - \alpha \frac{V_{\partial w}^c}{\sqrt{s_{\partial w}^c}} \\ b_{(t+1)} &= b_t - \alpha \frac{V_{\partial b}^c}{\sqrt{v_{\partial b}^c}} \end{aligned} \quad (\text{A.23})$$

A.2.2. Convolutional Neural Network

A convolutional neural network (CNN) is a popular model architecture for extracting features from the images. The below section goes through the key components of a convolutional neural network.

A.2.2.1. Convolution and its properties

Convolution layer extracts spatial information by applying kernel filters to the input image. Each convolutional layer has a collection of filters that convert the input into feature maps by learning complicated pattern during training. Convolution mathematically shares a similar structure as cross-correlation as given in the equation A.24. Convolution holds commutative property such that output remains the same when kernel and input are interchanged.

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k H[u, v] F[i - u, j - v] \quad (\text{A.24})$$

Unlike normal neural networks, weights of CNN are tied with each other enabling the parameter to share its local properties spatially. They also provide the benefit of translation equivariance where the change in the input would also shift the output accordingly. Convolution, on the other hand, is not equivariant for changes in size and rotation.

A.2.3. Pooling

Feature maps generated by the convolutional layer are passed through the pooling layer, usually after the non-linear activation function. The pooling layer captures global information without making the network bias towards the small non-rigid changes in the image. The pooling layer does not learn any parameters; instead, it decreases spatial resolution to smooth out the computational complexity of the subsequent layers. Commonly used pooling methods are max pooling and average pooling. Max pooling identifies the largest value in the provided patch of a feature map, whereas average pooling calculates the average value of the patch to downsample the given input as shown in figure A.7.

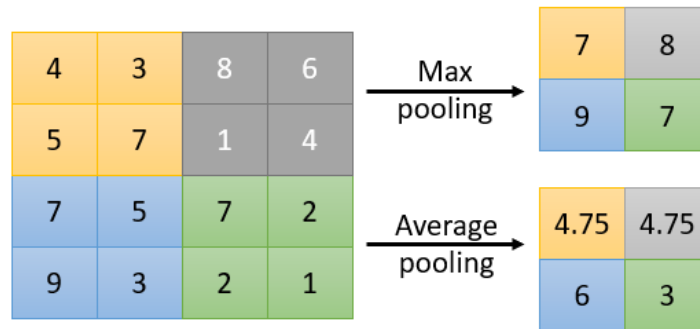


Figure A.7: Max and average pooling

A.2.4. Stride and Padding

The stride parameter controls the sliding interval of the kernel window over the input area. Applying Convolution over image usually reduces the dimension spatially but in some tasks it is essential to retain output size to establish dimensional coherence across the model structure. Padding is typically used to restore the size between input and output layers. Padding is applied around borders of input with a predefined value of zero.

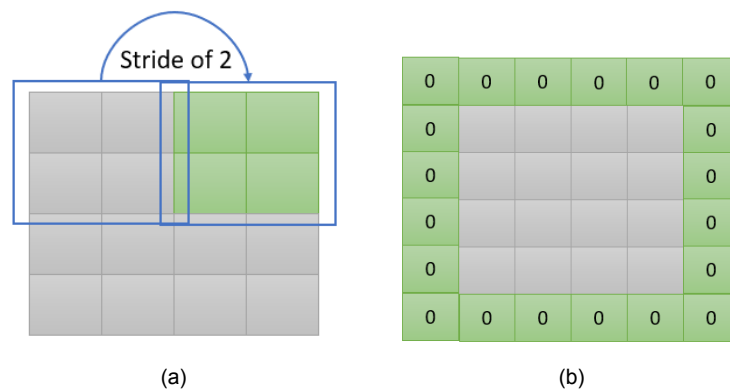


Figure A.8: Stride (a) and Pooling (b)

A.2.4.1. Upsampling

Padding could help retaining spatial resolution but the information at the boundaries is processed as zeros. On the other hand, upsampling techniques with transposed convolution proven to be an effective strategy for increasing the resolution of the features. Transposed convolution learns the weight and bias for upsampling the features which outperforms conventional interpolation algorithms such as bicubic and bilinear. The primary drawback is that separate weights have to be learnt during the training.

B

Appendix

B.1. Geometrical approach - Stage 2

The appendix B contains methodology studied during the phase of our project development but not included in the final pipeline. In section 3.2.3, the deep learning model detects the moving object in the scene. Based on the literature, the moving object in the scene could also be detected using geometrical approaches. One such geometrical approach, namely epipolar constraint, is studied in our project to classify potential dynamic objects into moving and non-moving entities.

Epipolar constraint is already associated in ORB-SLAM2 algorithm to find the dynamic points. The main difference between the epipolar constraint used in our implementation and ORB-SLAM2 lies with the handling of dynamic points. In ORB-SLAM2, the dynamic points could not associated with the individual object. Due to a lack of object boundary information, points associated with objects are not entirely categorized as dynamic would be included in the map. However, in our implementation, the points corresponding to the dynamic objects are classified based on the segmentation mask which ensures all the point associated with the object is removed during the SLAM process. Such an implementation would undoubtedly improve the efficiency of handling dynamic points present in the scene.

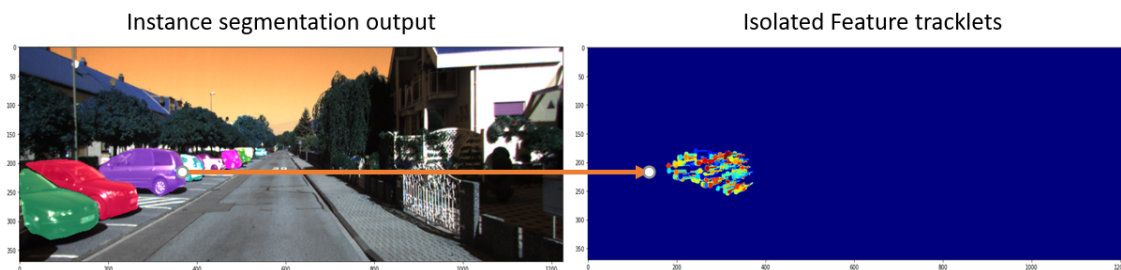


Figure B.1: Individual feature tracking using flow vectors

Lucas Kanade method of sparse optical flow is used to track the feature correspondence between two consecutive images. Let us consider the feature correspondence

points obtained from two images as f_t, f_{t+1} . The segmentation output is coupled to isolate features into the subset based on the instances, as shown in the figure B.1.

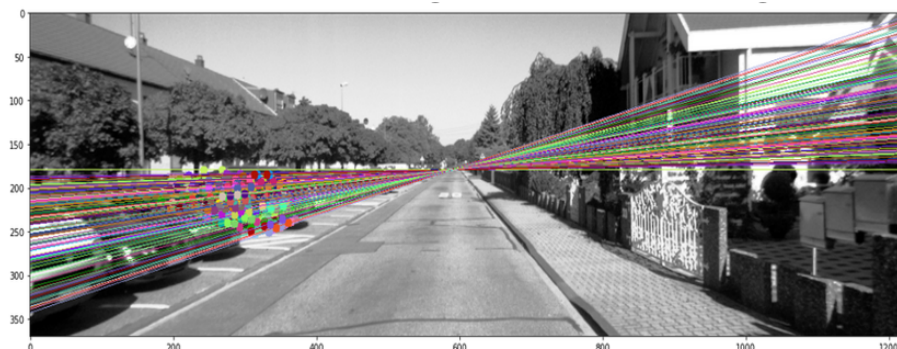


Figure B.2: Epipolar line with tracked features

The section A.1.3 suggest that an epipolar line could be formed based on the fundamental matrix (F) and feature point as $F * f_t$. Based on the epipolar constraint, feature correspondence associated with the static points in the next frame always lies closer to the epipolar line. If the perpendicular distance between the feature points and the epipolar line is higher than a threshold, the points are considered as dynamic. We consider an object with at least three dynamic points to be moving. The epipolar constraint governing the dynamic point detection is given by,

$$f_{t+1}^T \cdot F \cdot f_t = 0 \quad (\text{B.1})$$

The main challenges of using the geometrical approach involves the correct estimation of the fundamental matrix and mismatch of feature correspondence. Though the feature mismatch problem during our experiments is tackled using IOU and the multi-object tracking result, the fundamental matrix could not be estimated robustly. In the case of predominant camera rotation, the estimation of the fundamental matrix becomes erratic. Such an inaccurate fundamental matrix would result in outliers causing inefficient detection of dynamic points. Hence this approach is not implemented as part of our final pipeline.

B.2. Semi-dense Mapping

ORB-SLAM2 maps the features detected in multiple frames as the landmark in the 3D space. The sparsity of the map created by ORB-SLAM2 is one of its drawbacks when compared to direct SLAM approaches. This section explores the dense mapping technique adapted to the feature-based SLAM architecture to overcome the sparsity.

The current implementation is based on the research proposed in [113]. The main idea behind the semi-dense mapping lies in the estimation of inverse depth for the neighbouring pixel association. Feature pixels of keyframes with high gradient values are searched along the epipolar line to generate inverse depth hypotheses, probabilistically represented in the gaussian distribution. A cost function based on modulo

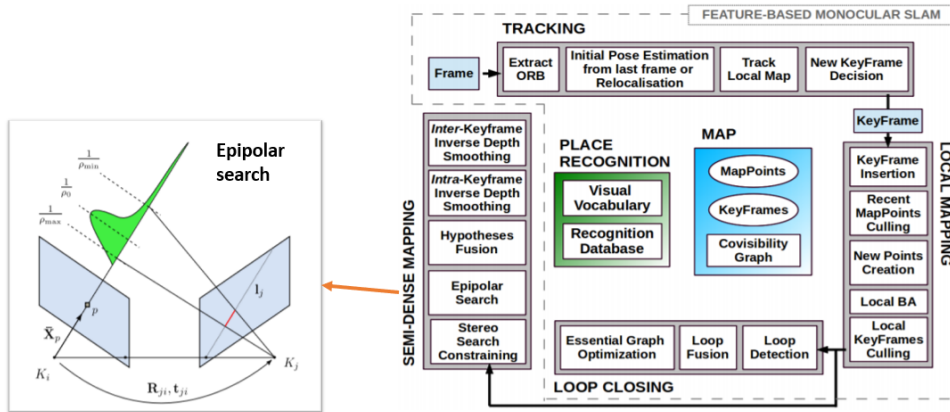


Figure B.3: Semi-dense mapping - ORB-SLAM2 implementation [113]

and orientation values is considered in addition to the intensity values to ensure robust estimation void of outliers. The search space of hypotheses is confined using the known depth estimated during tracking and local mapping. After the depth hypotheses, intra-keyframe and inter-keyframe depth checking and smoothening are performed to remove the outliers and provide smoothened edges. The qualitative result analysed in the paper suggest that reconstructed maps have better edges than LSD-SLAM.

The reconstructed semi-dense mapping of KITTI sequences is shown below,

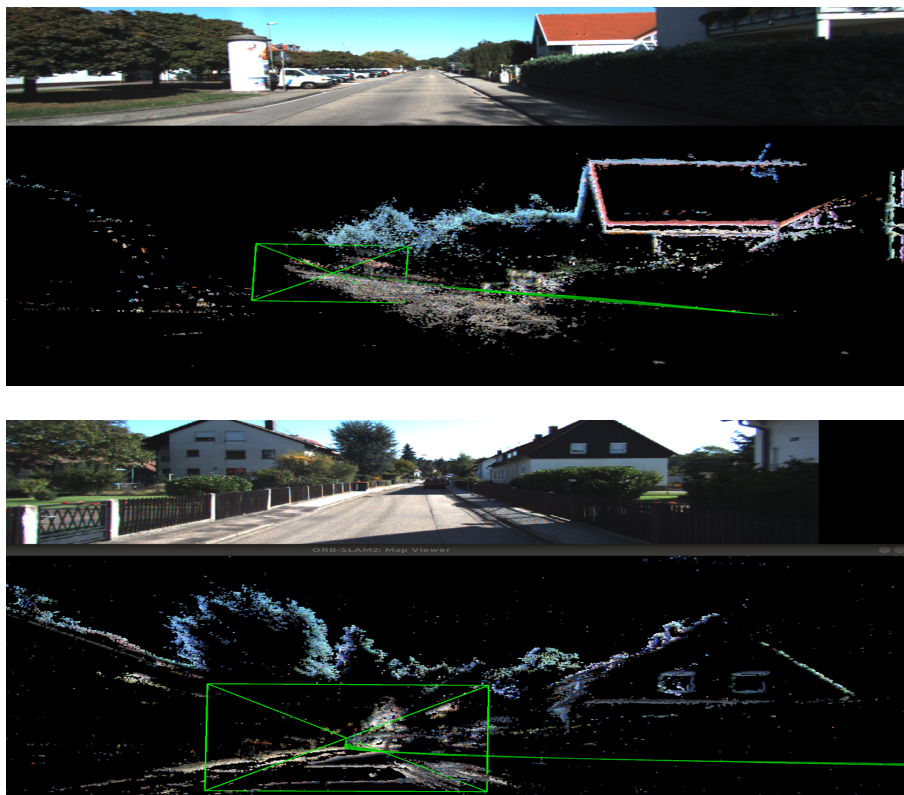


Figure B.4: Semi-dense mapping results

Bibliography

- [1] Giulio Reina, Andres Vargas, Keiji Nagatani, and Kazuya Yoshida. Adaptive kalman filtering for gps-based mobile robot localization. In *2007 IEEE International Workshop on Safety, Security and Rescue Robotics*, pages 1–6. IEEE, 2007.
- [2] Muhammad Sualeh and Gon-Woo Kim. Simultaneous localization and mapping in the epoch of semantics: a survey. *International Journal of Control, Automation and Systems*, 17(3):729–742, 2019.
- [3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on robotics*, 32(6):1309–1332, 2016.
- [4] Ioannis Kostavelis and Antonios Gasteratos. Semantic mapping for mobile robotics tasks: A survey. *Robotics and Autonomous Systems*, 66:86–103, 2015.
- [5] Berta Bescos, Carlos Campos, Juan D Tardós, and José Neira. Dynaslam ii: Tightly-coupled multi-object tracking and slam. *IEEE Robotics and Automation Letters*, 6(3):5191–5198, 2021.
- [6] Chieh-Chih Wang, Charles Thorpe, Sebastian Thrun, Martial Hebert, and Hugh Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *The International Journal of Robotics Research*, 26(9):889–916, 2007.
- [7] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):1–11, 2017.
- [8] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.
- [9] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.
- [10] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2017.

- [11] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.
- [12] Thomas Schöps, Jakob Engel, and Daniel Cremers. Semi-dense visual odometry for ar on a smartphone. In *2014 IEEE international symposium on mixed and augmented reality (ISMAR)*, pages 145–150. IEEE, 2014.
- [13] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.
- [14] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007.
- [15] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.
- [16] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.
- [17] Maksim Filipenko and Ilya Afanasyev. Comparison of various slam systems for mobile robot in an indoor environment. In *2018 International Conference on Intelligent Systems (IS)*, pages 400–407. IEEE, 2018.
- [18] Yin-Tien Wang, Ming-Chun Lin, and Rung-Chi Ju. Visual slam and moving-object detection for a small-size humanoid robot. *International Journal of Advanced Robotic Systems*, 7(2):13, 2010.
- [19] Yaser Sheikh, Omar Javed, and Takeo Kanade. Background subtraction for freely moving cameras. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1219–1225. IEEE, 2009.
- [20] Abhijit Kundu, K Madhava Krishna, and Jayanthi Sivaswamy. Moving object detection by multi-view geometric techniques from a single camera mounted robot. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4306–4312. IEEE, 2009.
- [21] Muhamad Risqi U Saputra, Andrew Markham, and Niki Trigoni. Visual slam and structure from motion in dynamic environments: A survey. *ACM Computing Surveys (CSUR)*, 51(2):1–36, 2018.
- [22] Gonzalo R Rodríguez-Canosa, Stephen Thomas, Jaime Del Cerro, Antonio Barrientos, and Bruce MacDonald. A real-time method to detect and track moving objects (datmo) from unmanned aerial vehicles (uavs) using a single camera. *Remote Sensing*, 4(4):1090–1111, 2012.

- [23] Boyoon Jung and Gaurav S Sukhatme. Detecting moving objects using a single camera on a mobile robot in an outdoor environment. In *International conference on intelligent autonomous systems*, pages 980–987. Citeseer, 2004.
- [24] Yuxiang Sun, Ming Liu, and Max Q-H Meng. Improving rgb-d slam in dynamic environments: A motion removal approach. *Robotics and Autonomous Systems*, 89:110–122, 2017.
- [25] Weichen Dai, Yu Zhang, Ping Li, Zheng Fang, and Sebastian Scherer. Rgb-d slam in dynamic environments using point correlations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.
- [26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [27] Gumin Jin, Xingjun Zhong, Shaoqing Fang, Xiangyu Deng, and Jianxun Li. Keyframe-based dynamic elimination slam system using yolo detection. In *International Conference on Intelligent Robotics and Applications*, pages 697–705. Springer, 2019.
- [28] Peiyu Guan, Zhiqiang Cao, Erkui Chen, Shuang Liang, Min Tan, and Junzhi Yu. A real-time semantic visual slam approach with points and objects. *International Journal of Advanced Robotic Systems*, 17(1):1729881420905443, 2020.
- [29] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *arXiv preprint arXiv:1606.02147*, 2016.
- [30] Zhuo Chen, Weimin Zhang, Fangxing Li, Yongliang Shi, Yang Wang, Fuyu Nie, Chi Zhu, and Qiang Huang. A research on the fusion of semantic segment network and slam. In *2019 IEEE International Conference on Advanced Robotics and its Social Impacts (ARSO)*, pages 304–309. IEEE, 2019.
- [31] Chao Yu, Zuxin Liu, Xin-Jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. Ds-slam: A semantic visual slam towards dynamic environments. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1168–1174. IEEE, 2018.
- [32] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.
- [33] Linyan Cui and Chaowei Ma. Sof-slam: A semantic visual slam for dynamic environments. *IEEE Access*, 7:166528–166539, 2019.
- [34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

- [35] Lili Zhao, Zhili Liu, Jianwen Chen, Weitong Cai, Wenyi Wang, and Liaoyuan Zeng. A compatible framework for rgb-d slam in dynamic scenes. *IEEE Access*, 7:75604–75614, 2019.
- [36] Berta Bescos, José M Fácil, Javier Civera, and José Neira. Dynaslam: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083, 2018.
- [37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.
- [38] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [39] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.
- [40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.
- [41] Lilian Weng. Object detection for dummies part 3: R-cnn family. *lilianweng.github.io/lil-log*, 2017.
- [42] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [43] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [44] Shijie Hao, Yuan Zhou, and Yanrong Guo. A brief survey on semantic segmentation with deep learning. *Neurocomputing*, 406:302–321, 2020.
- [45] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.
- [46] Olaf Ronneberger, Philipp Fischer, and Thomas 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.
- [47] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017.

- [48] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9157–9166, 2019.
- [49] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic, faster and stronger. *arXiv preprint arXiv:2003.10152*, 2020.
- [50] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. Solo: Segmenting objects by locations. In *European Conference on Computer Vision*, pages 649–665. Springer, 2020.
- [51] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.
- [52] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *arXiv preprint arXiv:1807.03247*, 2018.
- [53] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [54] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 240–248. Springer, 2017.
- [55] Zhilu Zhang and Mert R Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. In *32nd Conference on Neural Information Processing Systems (NeurIPS)*, 2018.
- [56] Yaoshiang Ho and Samuel Wookey. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access*, 8:4806–4813, 2019.
- [57] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 3551–3558, 2013.
- [58] Mennatullah Siam, Heba Mahgoub, Mohamed Zahran, Senthil Yogamani, Martin Jagersand, and Ahmad El-Sallab. Modnet: Moving object detection network with motion and appearance for autonomous driving. *arXiv preprint arXiv:1709.04821*, 2017.
- [59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

- [60] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.
- [61] Simon Baker, Daniel Scharstein, JP Lewis, Stefan Roth, Michael J Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International journal of computer vision*, 92(1):1–31, 2011.
- [62] Margarita Grinvald, Fadri Furrer, Tonci Novkovic, Jen Jen Chung, Cesar Cadena, Roland Siegwart, and Juan Nieto. Volumetric instance-aware semantic mapping and 3d object discovery. *IEEE Robotics and Automation Letters*, 4(3):3037–3044, 2019.
- [63] Martin Runz, Maud Buffier, and Lourdes Agapito. Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 10–20. IEEE, 2018.
- [64] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, and Taekyun Kim. Multiple object tracking: A literature review. *Artificial Intelligence*, page 103448, 2020.
- [65] Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. Mots: Multi-object tracking and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7942–7951, 2019.
- [66] Zhongdao Wang, Liang Zheng, Yixuan Liu, Yali Li, and Shengjin Wang. Towards real-time multi-object tracking. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 107–122. Springer, 2020.
- [67] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [68] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.
- [69] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 573–580. IEEE, 2012.
- [70] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

- [71] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [72] M Siam. Multi-task learning with motion and appearance.
- [73] Kentaro Wada. labelme: Image Polygonal Annotation with Python. <https://github.com/wkentaro/labelme>, 2016.
- [74] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04):376–380, 1991.
- [75] Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7244–7251. IEEE, 2018.
- [76] Hamid Rezaatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.
- [77] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019.
- [78] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.
- [79] Rishav Rishav, Ramy Battrawy, René Schuster, Oliver Wasenmüller, and Didier Stricker. Deeplidarflow: A deep learning architecture for scene flow estimation using monocular camera and sparse lidar. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10460–10467. IEEE, 2020.
- [80] Johan Vertens, Abhinav Valada, and Wolfram Burgard. Smsnet: Semantic motion segmentation using deep convolutional neural networks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 582–589. IEEE, 2017.
- [81] Hazem Rashed, Ahmad El Sallab, and Senthil Yogamani. Vm-modnet: Vehicle motion aware moving object detection for autonomous driving. *arXiv preprint arXiv:2104.10985*, 2021.
- [82] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

- [83] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.
- [84] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.
- [85] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *2011 IEEE international conference on robotics and automation*, pages 1–4. IEEE, 2011.
- [86] Armin Hornung, Kai M Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, 34(3):189–206, 2013.
- [87] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
- [88] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
- [89] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011.
- [90] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.
- [91] Robert C Bolles, H Harlyn Baker, and David H Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International journal of computer vision*, 1(1):7–55, 1987.
- [92] Richard I Hartley. In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 19(6):580–593, 1997.
- [93] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.
- [94] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.
- [95] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

- [96] Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.
- [97] Rainer Kümmeler, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.
- [98] G. Grisetti, R. Kümmeler, C. Stachniss, and W. Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2:31–43, 2010.
- [99] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.
- [100] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.
- [101] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8934–8943, 2018.
- [102] Liang Liu, Jiangning Zhang, Ruifei He, Yong Liu, Yabiao Wang, Ying Tai, Donghao Luo, Chengjie Wang, Jilin Li, and Feiyue Huang. Learning by analogy: Reliable supervision from transformations for unsupervised optical flow estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6489–6498, 2020.
- [103] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [104] S Agatonovic-Kustrin and Rosemary Beresford. Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22(5):717–727, 2000.
- [105] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [106] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.
- [107] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.
- [108] Josh Patterson and Adam Gibson. *Deep learning: A practitioner's approach*. "O'Reilly Media, Inc.", 2017.

-
- [109] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate bayesian inference. *arXiv preprint arXiv:1704.04289*, 2017.
- [110] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.
- [111] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8):2, 2012.
- [112] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [113] Raúl Mur-Artal and Juan D Tardós. Probabilistic semi-dense mapping from highly accurate feature-based monocular slam. In *Robotics: Science and Systems*, volume 2015. Rome, 2015.