FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Dynamics-Aware Visual Odometry System for Autonomous Driving

Nuno Manuel Canizes Ricardo

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Prof. Andry Maykol Pinto

Co-Supervisor: Prof. Aníbal Castilho Matos

July 27, 2022

Resumo

O setor automóvel encontra-se em rápida expansão, havendo um grande foco em aumentar as capacidades autónomas dos veículos. Este esforço envolve o desenvolvimento de métodos de autolocalização que sejam precisos e robustos. Estes são fulcrais em sistemas autónomos, servindo de base para outras tarefas como o planeamento de trajetória. A odometria visual tem um papel importante, sendo uma forma eficaz de determinar a odometria de um veículo, com precisão e robustez superior a outros sistemas mais tradicionais, como o GNSS. Esta dissertação tem como objetivo contribuir para o avanço de conhecimento na área da odometria visual, mais especificamente no contexto de condução autónoma. O trabalho realizado consiste em analisar o estado atual da odometria visual, tendo em conta diferentes tipos de sensores e técnicas, sendo estas tradicionais ou baseadas em inteligência artificial. Tendo o estado da arte em consideração, foi desenvolvido um novo sistema stereo de odometria visual, alavancando as potencialidades de arquiteturas de *deep learning* em áreas como a estimação de fluxo ótico e disparidade com métodos algébricos e modelos provados para estimação de odometria. Para além disto o sistema tem a especificidade de conter um módulo de deteção de objetos dinâmicos, para que estes não contribuam negativamente para as estimativas de odometria.

O sistema de odometria mostrou resultados promissores, atingindo erros da ordem dos 63 cm, no que toca a pose relativa, em certas sequências do *KITTI odometry benchmark*. A máscara de objetos dinâmicos também foi testada sob diversas condições atingindo um *dice-score* de 0.78. O algoritmo de odometria foi submetido a alguns cenários com abundância de objetos dinâmicos, sendo que a máscara conseguiu minimizar os impacto destes objetos nas estimativas de odometria. Também são de notar algumas das vantagens e desvantagens do sistema desenvolvido, obtendo os melhores resultados em ambientes urbanos e residenciais, mesmo com objetos dinâmicos. Em cenários de autoestrada ou com grande variabilidade de iluminação o desempenho tende a ser inferior. Os resultados foram comparados com outro método stereo de odometria visual do estado da arte. Este trabalho mostra que os próximos passos da odometria visual poderão passar pela integração de sistemas híbridos entre *deep learning* e métodos tradicionais.

Abstract

The automotive industry is expanding rapidly, and there is a strong focus on increasing the autonomous capabilities of vehicles. This effort involves the development of accurate and robust self-localization methods. These are central to autonomous systems and serve as the basis for other tasks such as trajectory planning. Visual odometry plays an important role, being an effective way to determine the ego-motion of a vehicle, with accuracy and robustness superior to other more traditional systems, such as GNSS. This dissertation aims to contribute to the advancement of knowledge in the area of visual odometry, more specifically in the context of autonomous driving. The developed work consists of analyzing the current state of visual odometry, taking into account different types of sensors and techniques, these being knowledge or learning-based. Taking the state-of-the-art into consideration, a new stereo visual odometry system was developed, leveraging the potential of deep learning architectures in areas such as optical flow and disparity with algebraic methods and proven models for ego-motion estimation. In addition, the system has the specificity of containing a dynamic object detection module, so that these do not contribute negatively to the odometry estimates.

The odometry system showed promising results, reaching errors in the order of 63 cm, regarding the relative pose, in certain sequences of the KITTI odometry benchmark. The dynamic object mask was also tested under various conditions reaching a dice-score of 0.78. The odometry algorithm was subjected to some scenarios with an abundance of dynamic objects, in these cases the mask was able to minimize the impact of these objects on the ego-motion estimates. Also noteworthy are some of the advantages and disadvantages of the developed system, obtaining the best results in urban and residential environments, even with dynamic objects. In motorway scenarios and with high lighting variability, performance tends to be lower. The results were compared with another state-of-the-art stereo visual odometry method. This work shows that the next steps in visual odometry may involve the integration of hybrid systems between deep learning and traditional methods.

Agradecimentos

Ao meu orientador Prof. Andry Pinto pela confiança, dedicação e orientação durante uma grande parte do meu percurso académico. Todo o seu apoio levou-me a alcançar objetivos que não pensava possíveis. À minha co-orientadora Maria Pereira por toda ajuda, disponibilidade e partilha de conhecimento que tornou possível a conclusão deste longo e árduo trabalho.

A todos os meus amigos, especialmente ao Lucas, David e André pela entre-ajuda e momentos partilhados que ajudaram a tornar estes cinco anos bem mais fáceis.

A toda a minha família que sempre me apoiou incondicionalmente. Sempre puseram a minha educação em primeiro lugar, sem eles não seria possível.

À Ana que me acompanhou durante estes cinco anos, pelas vezes que me aguentou menos bem disposto quando as coisas não corriam bem. Mas acima de tudo pelos bons momentos que me ajudaram a terminar o curso.

Obrigado.

Nuno Ricardo

Official Acknowledgments

This work was carried out with the support of the host institution: INESC TEC and supervised at the institution by: Maria Inês Pereira.

This thesis was supported by the European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project nº 047264; Funding Reference: POCI-01-0247-FEDER-047264].

"The way to succeed is to double your failure rate."

Thomas J. Watson

Contents

List of Figures

List of Tables

Acronyms and Symbols

2D	Two-dimensional
3D	Three-dimensional
AEE	Average End-point Error
ATE	Absolute Trajectory Error
AV	Autonomous Vehicle
CBAM	Convolutional Attention Module
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DoF	Degrees of Freedom
EPE	End-Point Error
FN	False Negative
FOV	Field Of View
FP	False Positive
GNSS	Global Navigation Satellite System
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
GT	Ground Truth
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
IoU	Intersection over Union
LiDAR	Light Detection and Ranging
MOS	Moving Object Segmentation
NN	Neural Network
PCL	Point Cloud Library
RANSAC	RANdom SAmple Consensus
RGB	Red Green Blue
RGB-D	Red Green Blue-Depth
RMSE	Root-Mean-Square Error
RNN	Recurrent Neural Network
ROS	Robot Operating System
RPE	Relative Pose Error
SAE	Society of Automotive Engineers
SfM	Structure from Motion
SLAM	Simultaneous Location and Mapping
SSD	Single Shot Detector
SVD	Single Value Decomposition
TN	True Negative
TP	True Positive
UAV	Unmanned Aerial Vehicle
VO	Visual Odometry

Chapter 1

Introduction

1.1 Context and Motivation

The automotive industry is changing. Step by step, manufacturers are improving the autonomous capabilities of new vehicles. The Society of Automotive Engineers (SAE) International distinguishes six levels for driving automation systems: from level zero, where the human driver takes full control of the vehicle, to level five, where the autonomous system does not require the supervision or control from the human driver, and it is capable of controlling the vehicle in all driving conditions. The most advanced vehicles available to consumers today only offer level two, where the system provides steering and brake/acceleration assists to the driver, but the human must supervise and provide inputs if the situation requires it.

Studies from Panagiotopoulos *et al.* [1] show that a large percentage of drivers are somewhat or extremely likely to have or use Autonomous Vehicles (AVs) when they are on the market, but are also largely concerned about system security, data privacy, and the adaptation curve. Nonetheless, manufacturers are racing to improve the autonomous driving capabilities of their vehicles. As exposed by Barabás *et al.* [2], this evolution has several benefits but also faces some controversial aspects. It is almost a common opinion that the introduction of AVs will improve traffic and environmental issues. This is the case because AVs will have the ability to communicate with each other, being able to optimize braking/accelerating moments using platooning methods and avoid heavily congested areas such as city centers. These autonomous systems also raise several ethical and safety problems. Resuming the work of Maurer *et al.* [3], an AV needs to make decisions. For example, when faced with the moral dilemma to save the occupants inside instead of the pedestrian in a certain crash scenario, the system will be taking control of human lives. In the case of fatalities who would be the culprit? This is a moral dilemma that is discussed since classical times and has no clear logical answer. Therefore, legislation has to accompany the evolution of AVs in this manner.

Unanticipated scenarios are another possible safety hazard. When unexpected phenomena happen, humans rely on common sense. This could be when the road is flooded or a deer jumps

in front of the vehicle. Unfortunately, this is not something that can be easily introduced in an autonomous system. With the rise of artificial intelligence systems may have the ability to generalize and respond in a safe way against some of these hazards, but are still far behind common human reasoning.

Odometry and trajectory estimation systems have a central role when safety is concerned. To make the right decisions in critical situations the AV needs to have a clear perception of the environment. This involves two major tasks: to be aware of moving agents around it such as cars, trucks or pedestrians and to know its location both in relation to static structures like buildings and to dynamic objects. Odometry has noticed relevant research advancements in the past few years. Some reasons might be the large advancements in artificial intelligence, sensor technologies and computing power but also the big push that vehicle manufacturers are doing in order to further advance the capabilities of AVs. One major inconvenience that odometry systems deal with is the existence of dynamic objects in the environment. If not taken into account, pose and trajectory estimations may become corrupted by assuming that a moving object is static and therefore basing calculations on wrong assumptions. This is why the detection, segmentation and removal of dynamic objects is crucial in the context of visual odometry and autonomous driving.

Thus technological advancement is needed to further improve the capabilities of these systems, not only in terms of safety but also to further improve trajectory estimation. This thesis focuses on that, by proposing a dynamic-aware visual odometry system that can mitigate safety concerns by knowing the whereabouts of moving agents, and improve on current visual and laser odometry systems that are severely affected by dynamic objects in their surroundings.

1.2 Objectives

This dissertation aims to address the existence of dynamic elements in the environment, that introduce errors in visual odometry measurements. The goal is to develop a visual odometry system, based on a stereo camera sensor, that can segment dynamic objects and thus reduce their impact on trajectory estimation. The main objectives of this thesis are:

- Development of a method for dynamic object segmentation using cameras and depth sensors mounted on a moving vehicle.
- Implementation of a visual odometry system with integration of the dynamic object segmentation module.
- Evaluation of the dynamic object segmentation method in a wide variety of scenarios, containing a large amount of moving objects.
- Testing of the visual odometry system in a variety of static and dynamic environments using real-world data.

1.3 Contributions

This MSc dissertation thesis was developed within project THEIA- Automated Perception Driving, currently being developed between Faculdade de Engenharia da Universidade do Porto and Bosch Portugal. Some efforts were made to contribute to this project:

- Publication of a journal article on IEEE Access entitled "A Practical Survey on Visual Odometry for Autonomous Driving in Challenging Scenarios and Conditions" containing a literature review on visual odometry and a benchmark of some methods on challenging scenarios.

1.4 Document Structure

Adding to this introductory chapter this document has four other chapters. In chapter 2 the state-of-the-art of visual and point cloud based odometry as well as the detection of dynamic objects is discussed. In section 2.1 an overview of the basic concepts, as well, as some limitations are presented regarding odometry and dynamic object detection. Sections 2.2 and 2.4 show some recent works that are analyzed and compared. Finally section 2.5 contains a short critical review regarding the state-of-the-art approaches. Chapter 3 describes the faced problem, as well as a detailed description of the implemented methodology. In Chapter 4 a module to module, as well as a general evaluation of the system is performed under a variety of conditions. Chapter 5 contains the main conclusions as well as future work.

Chapter 2

State of the Art

Ego-motion estimation is one of the biggest challenges in the area of robotics, and it is easy to see why: to move, avoid collisions and perform complex missions, a robotic system needs a way to estimate its current position and orientation. The use of multiple sensors to perform this task is defined as odometry and it is of special interest in growing areas such as autonomous driving where the ability of self-localization is extremely important. Odometry can be employed in several ways, using wheel speed encoders, radars, LiDARs, cameras or a mixture of various sensors. In the next sections, visual and laser-based odometry methods will be explored in depth.

2.1 Introduction

The term Visual Odometry (VO) was introduced by Nister *et al.* [4] and it is applicable when cameras are used to estimate the ego-motion of an agent, where typically it moves through the environment taking sequential images with a rigidly attached camera at discrete time instants. VO can be seen as a particular case of the Structure from Motion (SfM) problem, where the objective is to recover relative camera poses and 3D structures from several projections into a series of images, taken from different perspectives [5]. This means VO recovers the path incrementally, pose by pose. As summarized by Scaramuzza [6] and Alkendi [7], there are three standard VO motion estimation algorithms: 2D to 2D, 3D to 2D, and 3D to 3D. These methods are used to compute the transformation matrix between two consecutive frames:

- **2D to 2D**- the transformation matrix is derived using the essential matrix that is defined by the geometric relationship between two consecutive frames and computed from the 2D feature correspondences using the epipolar constraint as illustrated in Figure 2.1;
- **3D to 2D**- the transformation matrix is computed by minimizing the 2D to 3D re-projection error from 2D and 3D correspondences;
- **3D to 3D**- the transformation matrix is computed by comparing two sets of three-dimensional correspondences, by triangulating the matched points for each stereo pair;

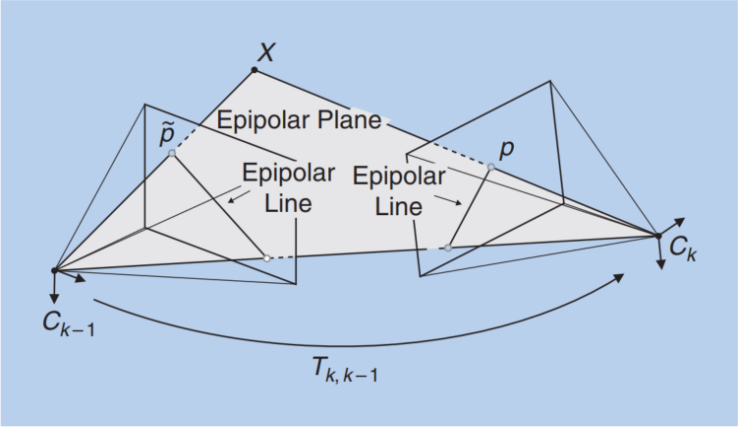


Figure 2.1: Illustration of epipolar constraint [6]. Where C_k and C_{k-1} represent the camera centers for two consecutive frames, p̃ and p are the projections of X on each frame and $T_{k,k-1}$ is the geometric transformation between the frames.

The epipolar constraint is defined by the following equation:

$$\tilde{p}^T F p = 0 \tag{2.1}$$

where p̃ and p are corresponding points in consecutive frames in homogeneous coordinates and F is the fundamental matrix that contains the translation and rotation between frames as well as the camera intrinsic parameters. After obtaining the transformation matrix, an iterative refinement process is carried over the last poses to obtain a better estimate of the local trajectory, which is done by minimizing the sum of the squared re-projection errors of the reconstructed 3D points over a window of past frames, this operation is called optimization/bundle adjustment:

$$argmin_{X^i,C_k} \sum_{i,k} ||p_k^i - g(X^i,C_k)||^2 \tag{2.2}$$

where p_k^i is the ith image point of the landmark X^i seen in the kth image and $g(X^i,C_k)$ is the image reprojection in camera pose C_k. VO can be employed using various techniques, depending if all raw data in an image is used to estimate the pose (appearance-based), or only specific and distinguishable features, (feature-based).

Appearance-based methods use the raw data from the images in terms of pixels, by analyzing the change in appearance between consecutive frames, i.e. the intensity of image pixels. An optical flow algorithm uses geometrical information in the consecutive camera frames and computes a 2D displacement vector that represents the movement, this is typically done by minimizing the photometric error. Optical flow algorithms that are used can be considered as dense if they use all image data or sparse if only selected pixels are used. This type of algorithm is good at minimizing aliasing issues when there are multiple similar patterns, and compared to feature-based methods provides accuracy and robustness in low texture and low visibility environments.

Feature-based methods are the other type of conventional VO. The first step involves the pre-processing of the images and detecting key distinctive information, using corner detectors such as the Harris corner detector, or local feature detectors such as SURF [8] or SIFT, across several camera frames. The next step is to match these features between images, which can be accomplished using the unique descriptors that feature detectors such as SIFT already provide. The final step is an optimization process that minimizes the geometric error between frames and calculates the transformation matrix between consecutive poses. The major advantage of this type of approach is the high robustness against geometrical distortions. The general pipeline for this type of approach is presented in Figure 2.2.

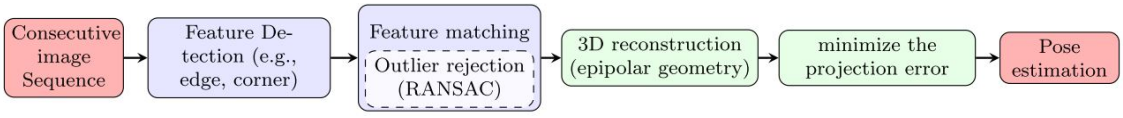


Figure 2.2: General pipeline for feature-based approaches [9].

Apart from traditional methods, deep learning techniques are becoming more and more prevalent, this type of approach has already proven to be effective in other areas of computer vision such as object detection and segmentation, for example, the well-known Convolutional Neural Networks (CNNs). These methods train neural networks with large amounts of data and then employ them. They provide accurate motion estimation as well as faster processing speed in relation to the latter methods, another advantage is that the camera calibration procedure is not necessary because camera parameters are not needed with this type of approach.

As seen above the explained methods use sequences of images taken by cameras to estimate the motion and pose of an agent. More recently, with the development of new sensors, ego-motion estimation methods also evolved to accompany the trends in technology. The use of Light Detection and Ranging (LiDAR) is becoming more and more prevalent since scans provide an accurate 3D representation of the environment. Therefore the basics of such algorithms will also be presented.

Before understanding LiDAR-based odometry algorithms, it is important to become familiar with some basic concepts. As explained by Roriz *et al.* [10] this type of sensor works by calculating the time of flight of a laser ray, based on the formula:

$$R = 0.5ct \tag{2.3}$$

Where R is the range, c is the speed of light and t is the round trip delay. To create a 3D point cloud of the environment it is necessary to do this calculation for a large number of laser rays. For this there are several types of imaging systems, the most used in current autonomous vehicles are the are rotor based mechanical LiDARs that work by physically spinning the sensor and scanning the environment in parts, normally these types of sensors have a 360-degree horizontal field of view with the vertical field of view varying from model to model.

LiDAR odometry methods are becoming increasingly popular, evidence to support this is that all top-scoring odometry algorithms on the KITTI autonomous driving benchmark are LiDAR-based. According to Jonnavithula [11], there are 5 basic steps for general laser odometry algorithms: (1) pre-processing, (2) feature extraction, (3) correspondence searching, (4) transformation estimation, and (5) post-processing. In pre-processing, point clouds are typically organized into more convenient forms, this being segmenting the point cloud or projecting it into a 2D representation, some processes like ground and dynamic object removal are also applied in this step. Feature extraction is done much like in VO using extractors like SURF [8] and SIFT, where key points or features are extracted. In terms of correspondence searching, matches are found to create correspondences between point clouds. In the fourth step, the transformation between consecutive poses is estimated and the ego-motion is recovered. Post-processing depends on the method used, but normally it is referred to some type of iterative refinement process.

Elhousni *et al.* [12] helps to define the three main types of approaches used in LiDAR odometry: registration-based, feature-based, and deep learning-based methods: Registration methods take two point clouds and try to align them with reference to the same frame, which makes it possible to retrieve the transformation between the two point clouds in terms of translation and rotation, an example is exposed in Figure 2.3. Registration methods can be used in two ways: by aligning newly formed point clouds with already built maps to locate the agent, or combining consecutive scans sequentially to obtain the ego-motion estimation; feature-based methods de-

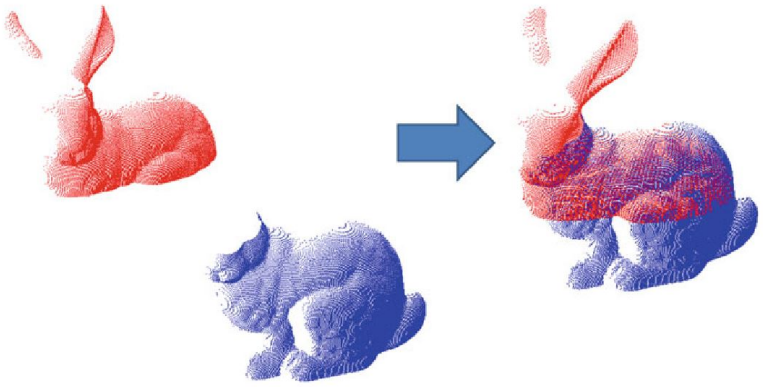


Figure 2.3: Registration of two point clouds [13].

tect points of interest, these are recognizable and consistent in time and space. In point clouds, features are normally described by a vector named feature descriptor. Using this vector it is possible to form feature correspondences between point clouds. With enough matches, the transform matrix between poses can be obtained. Deep learning methods in point cloud-based odometry are similar to the already covered in visual odometry, they provide many of the same upsides and work in the same basic way. The large computing power of neural networks is of high relevancy when working with large unorganized point clouds. Deep learning methods can estimate the ego-motion of an agent, while not using traditional approaches, taking the scanned point clouds as input and

retrieving the odometry, or can be used to replace certain parts of the traditional pipelines, the same could be said in visual odometry.

All of the presented approaches have several limitations that are difficult to surpass, no single method or odometry technique can be considered as the better one, on the contrary, combining several approaches could be the solution to minimize the downsides of each method. One major limitation when talking about VO is the weather and meteorological conditions, when the visibility is low or the sensors are partially blocked, for example in the presence of fog, heavy rain, or snow VO algorithms tend to perform poorly, there are ways to mitigate this by cleaning the lenses or using other sensors to aid the camera but it continues to be a hard problem to solve. Lighting conditions are also very important, in low-light scenarios VO methods usually under perform, as explained before in low-lighting conditions appearance-based methods tend to be more accurate than feature-based methods, because of the difficulty of detecting robust features when the image texture is low.

Point cloud-based odometry techniques that use LiDAR sensors are not affected by lighting conditions, another advantage over cameras is that they also measure the range to objects directly from world coordinates. On the other hand, LiDARs are also extremely affected by adverse weather conditions as scans are affected by atmospheric conditions. Carballo *et al.* [14] tested twelve different LiDAR models in adverse weather conditions like heavy fog and rain, proving a bad performance, for example in heavy rain, the laser beams reflected on water drops and created "rain pillars" in the scans, Figure 2.4. These types of sensors are also heavily affected by reflectivity, having difficulties detecting objects with low reflective properties, such as glass.

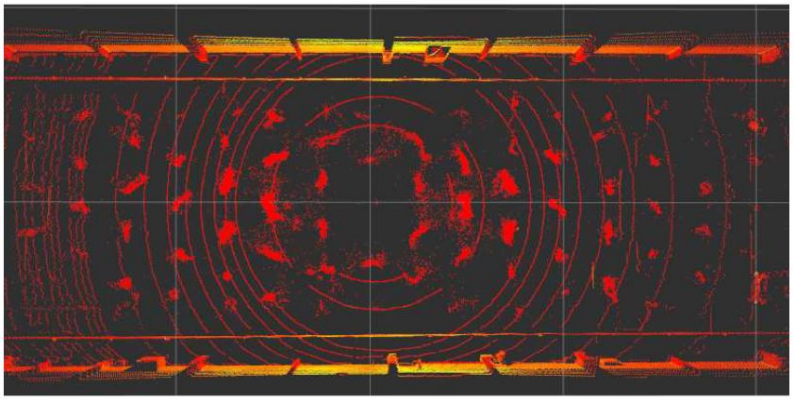


Figure 2.4: LiDAR scan (top view) taken under heavy rain. The lasers reflect on rain drops creating "rain pillars" [14].

Limitations in terms of computing power and time should also be addressed. This is especially important in the context of autonomous driving where the estimations should be provided at a high enough rate to meet the demanding time constraints associated with AVs. If the onboard computing unit of the vehicle is not powerful enough or the algorithm is inefficient these could have serious consequences, especially as many authors use high-performance hardware to develop their works, and these are not normally equipped in vehicles. In VO computing times are especially

important if the AV has several cameras and in term, these have high resolutions, the amount of data to process can be unmanageable, one way to reduce the computational burden could involve carefully selecting features, instead of working with the entire raw image, as exposed before. In point cloud techniques, the problems are the same with the added cost that many times point clouds are unordered, looking for points in these clouds or attempting to order them can be a time costly task. One way to order and store point clouds efficiently is through the use of an octree structure, which is a tree-based structure where each node has eight children, or in a K-D tree, that allows fast multidimensional searches.

Dynamic object detection is of great interest while discussing visual and laser odometry especially in the context of autonomous driving. While some authors take dynamic objects into account, most do not. This means that it is assumed that the world is static, which has major implications when estimating the ego-motion of a vehicle. Dynamic objects, when not accounted for, introduce errors in trajectory estimation. If there are only static objects in the environment, odometry is a straightforward task according to the chosen method and the sensors that are used (LiDAR, cameras). But, if moving objects are considered static, the calculation will be based on wrong assumptions and the pose estimation will not be accurate. Some recent efforts were made in the direction of detecting dynamic objects in the autonomous driving scenario, which can be especially difficult. With this in mind, the basic concepts of motion analysis and the detection of moving objects in the context of autonomous driving will be exposed.

Motion analysis has been an important and challenging area of computer vision for a large number of years with high applicability in a vast number of research fields. The concept of optical flow is defined as the motion of an object between two consecutive frames caused by the relative motion between the object and the camera. When the motion is evaluated in the 3D space, for example using a point cloud representation, it normally is referred to as scene flow, which in term judges the displacement of 3D points over time. For clarification, it is possible to determine the movement of objects in the three-dimensional space using stereo cameras, such methods are normally put in the category of 2.5D scene flow instead of 3D scene flow.

Despite this, according to Zhai *et al.* [15], all motion analysis works can be classified as one of three categories: knowledge-driven, data-driven, and hybrid-driven. Where knowledge based-methods can be considered as traditional and work on some prior knowledge which is used to build an algorithm that is able to model the spatio-temporal aspects of objects, data-driven methods rely on the recent advancements in neural networks and deep learning, large amounts of data are used to train a network that when fully trained can be employed in an operational scenario. Hybrid-driven methods, as the name suggests, are a combination of both. Lately, the latter two methods have been receiving a large amount of research resulting in a variety of new and interesting work.

Traditional optical and scene flow normally try to minimize a cost function with two terms. The data term incentives the alignment of similar areas across consecutive images or point clouds. The smoothness term imposes limitations on the probability of motion. This type of approach will be analyzed in more detail in section 2.4. Specifically in optical flow the brightness constancy

constraint is assumed:

$$I(x,y,t) = I(x+\delta_x, y+\delta_y, t+\delta_t) \tag{2.4}$$

Where $I(x,y,t)$ is the image intensity at coordinates (x,y) at time t. It is assumed that a pixel has a movement of (δ_x, δ_y) during a time interval of δ_t. The assumption made in equation 2.4 has several limitations: large displacement, occlusions, and illumination variations. If objects move at a very fast speed relative to the sensor, their representations in consecutive frames will tend to be far apart and their appearance might change drastically, because of the different relative pose between the sensor and the object, making it difficult to estimate the motion. Occlusions present another obstacle, if the moving object is occluded by a few moments, the algorithm will not be able to estimate its motion. This is very obvious in the autonomous driving scenario, if a car moves behind a truck and it's no longer visible, common human reasoning indicates the driver that the car is behind the truck and might show up again at any time. Well as mentioned before this type of common sense is only intrinsic to humans and very hard to implement. Lighting variations are also complicated, as shown in equation 2.4 brightness consistency is assumed across frames,variations can drastically change the appearance, making it difficult to infer similarities between frames. Despite this, deep learning state-of-the-art alternatives are comparable in terms of performance while having lower inference times. It is also important to note that the task of detecting dynamic objects can be especially complicated in the area of autonomous driving as the sensor is mounted on a moving vehicle, therefore the detected motion has two components, the ego-motion (movement of the sensor) and the object's motion, it is necessary two separate these components to robustly detect moving objects, one way to do this might be using odometry algorithms to infer the ego-motion and then extrapolate the detected object's movement by subtracting it.

A typical motion estimation done by an optical flow algorithm is depicted in Figure 2.5, where the color indicates the direction of movement of the pixels and the intensity represents the relative velocity.

2.2 Visual Odometry

2.2.1 Visual Odometry: Appearance-based methods

The work of Engel *et al.* [17], published in 2017 can be considered a landmark paper in the area. This paper proposes a direct sparse approach to monocular visual odometry. The basic concept of this proposal consists on continuously optimizing the photometric error over a window of recent frames taking into account a calibrated model for the image formation. This has two steps: the photometric calibration and the geometric calibration that assumes the pinhole model to obtain the camera's intrinsic parameters. The photometric error is defined as the weighted Single Shot Detector (SSD) over a small neighborhood. Bundle adjustment is done by optimizing the total error over a sliding window using the Gauss-Newton method. After establishing the latter definitions, the front end the algorithm needs to be discussed. It first determines which frames should be used, and which points in selected frames should be taken into account to perform

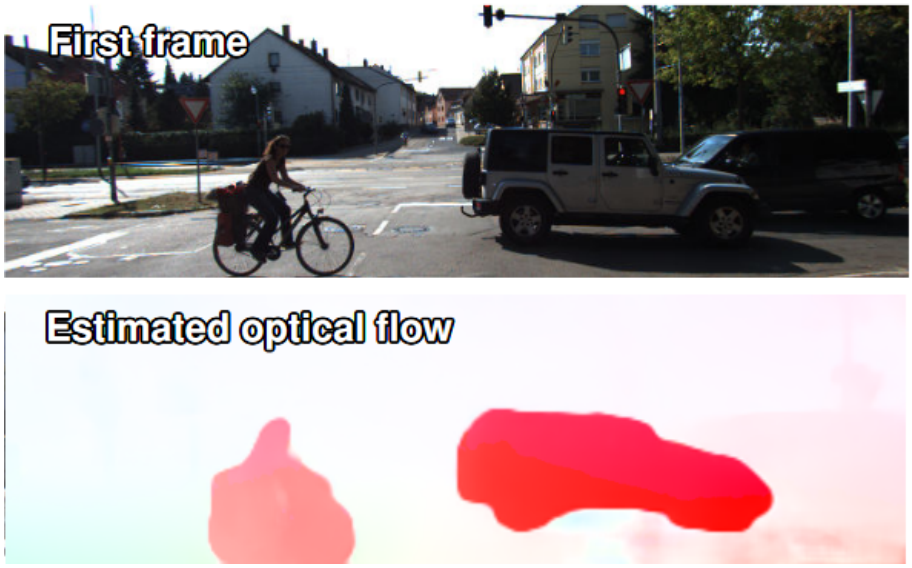


Figure 2.5: Typical optical flow representation, bottom image: color indicates direction of movement and brightness indicates relative velocity [16].

calculations. These points should be well distributed and have a high image gradient concerning their surroundings. Finally, selected points are tracked using a discrete search along the epipolar line, minimizing the photometric error. Because DSO is such an important work many authors use the same ideas in their works. For example, the work of Kim *et al.* [18] proposes a Background Model-based dense Visual-Odometry (BaMVO) algorithm, using an RGB-D sensor for robust navigation in dynamic environments. The background model is considered to reduce influences of the moving objects in the visual odometry calculation. This algorithm has two main sections that are interconnected: the background model and the ego-motion estimators. The background model is calculated by accumulating depth differences between frames. Because the sensor is moving, the frames are acquired from different viewpoints, so warping is applied to simulate that all frames are in the same viewpoint. Then depth images are calculated and the background model is obtained. For the ego-motion estimation, the sensor model is computed to increase the robustness and energy-based dense visual odometry is used. The algorithm uses the background model images, and determines the odometry by minimizing an energy function that maximizes the photo consistency between images and minimizes the effect of moving objects. A schematic overview of the algorithm is presented in Figure 2.6. This method already presents some ideas in the area of dynamic object detection in the area of autonomous vehicles that will be discussed in detail in section 2.4.

BaMVO was run against DVO using the TUM RGB-D datasets, comparing the root-mean-square-error (RMSE) of translational and rotational drift along with run times. In static environments BaMVO does not present major advantages over DVO in terms of drift errors while having longer runtimes. In dynamic environmets BaMVO excels over the more classical method since it has lower translation and rotation RMSEs, 0.2326 m/s and 4.3911 deg/s respectivly against DSO's

0.4360 m/s and 7.6669 deg/s.

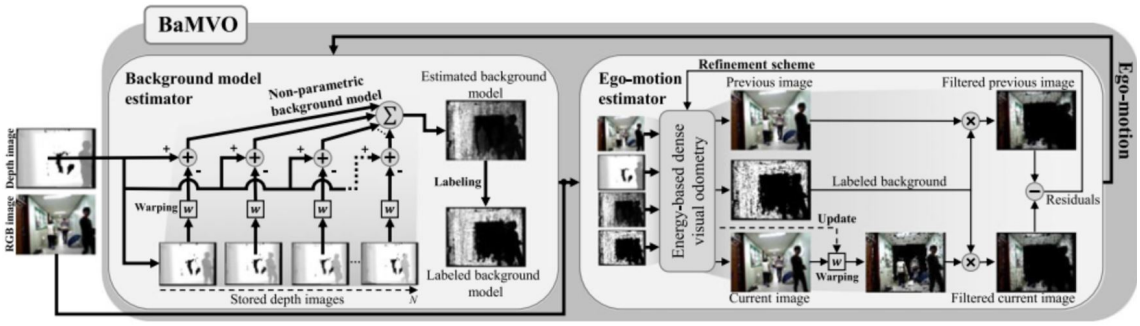


Figure 2.6: Schematic overview of BaMVO [18]. The depth images are warped using pre-computed ego-motions and used to create a non-parametric background model. To compute the ego-motion the dense information in the RGB image is also used along with the background model and the energy-based method is applied.

DV-LOAM [19] is another recent method that implements a depth-enhanced direct visual odometry algorithm, combined with a LiDAR mapping module to estimate ego-motion and generate a map. The system consists of a front-end and a back-end module. The front-end module takes synchronized images and LiDAR scans, from which the motion is estimated with a direct frame-to-frame tracking method without using any visual features. This calculation is based on the previously covered DSO [17] method. The slight variation in this case is that LiDAR points are carefully chosen through salient point detection [20], which consists in uniformly sampling the point cloud and choosing the points with the largest gradient, to estimate the depth. Afterwards a sliding window optimization module and a LiDAR based scan-to-map method are used to refine the previous estimation. The back-end module includes a loop closure method that relies on the constructed map to refine the ego-motion estimation when loop closure is detected. It is also important to point out that this work is very similar to the method previously proposed by Shin *et al.* [21] that is DVL-SLAM, that combines the sparse depth measurement of light detection and ranging (LiDAR) with a monocular camera into a SLAM (Simultaneous Localization and Mapping) algorithm. Tests on the KITTI benchmark achieved an average frequency of 6Hz, while being able to detect every loop-closure. This method achieved an average translational error of 1.11% and a rotational error of 0.005 deg/m.

2.2.2 Visual Odometry: Feature-based methods

ORB-SLAM 2 [22] and its previous iteration (ORB-SLAM [23]) are cornerstones in the area of VO, as the name suggests this work uses the ORB feature detector [24] as a base to a SLAM algorithm that not only calculates real-time ego-motion but also builds a map of the environment, which differs from pure odometry algorithms that only estimate the pose of the sensor on a frame by frame basis. This system has 3 parallel threads. The tracking where the ORB feature detector is used on sequential frames, these features are then matched across the sequential images and the

pose can be retrieved by motion estimation. Figure 2.7 shows the matching using ORB features across two real world images, where bundle adjustment is also used for optimization purposes. The second thread builds the local map once there is an initial estimation of the camera pose and feature matching. The third and last thread belongs to loop closure. This method allows to remove integration errors that are caused by the frame to frame pose estimation, by comparing current frames with the built map, and if similarities are detected the pose estimation can be refined. It is important to point out that this system fills some of the major gaps in pure odometry algorithms, such as accumulating errors, by creating a local map. During tests it was concluded that the computing time was low enough to work in real-time, the average translational and rotational error measured with the KITTI dataset are 1.15% and 0.0027 deg/m.

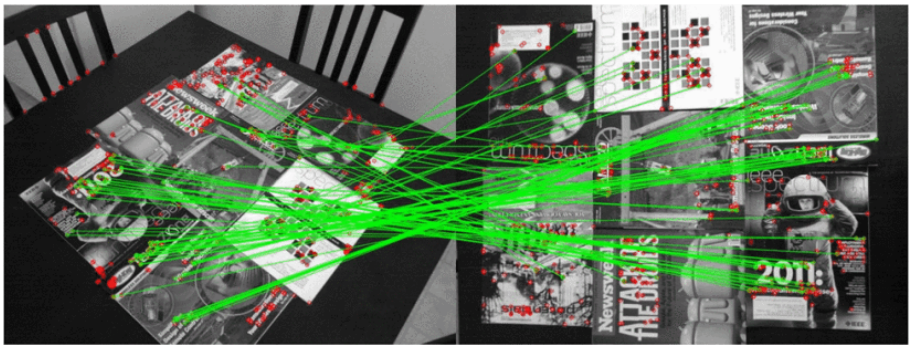


Figure 2.7: Matching two images using ORB features [24].

SOFT-SLAM [25] is another work of great relevance in the context of feature-based visual odometry. It was designed to be used in the autonomous navigation of unmanned aerial vehicles (UAVs) to be computationally efficient and highly reliable. Being a SLAM algorithm, it is composed of 2 main threads: odometry and mapping, and the general structure of the algorithm is exposed in Figure 2.8. The odometry thread takes a pair of stereo images as input and then detects high-quality features by applying corner and blob masks on the gradient image followed by non-maximum suppression. An inertial measurement unit (IMU) helps to predict the relative displacement between frames to better define the search location to do feature matching. This matching is done using the sum of absolute differences of patches of the gradient image. The output is a sparse feature set where RANSAC optimization is applied resulting in a set of inlier points that are used in a Gauss-Newton algorithm to estimate the orientation and translation. The mapping thread uses the frames where the features were detected and checks if they should be taken into consideration or discarded depending on distances to the last stored keyframe. If positive, the frames are checked for loop-closure that, if detected, is used to remove odometry drift errors. When first published, this method achieved state-of-the-art performance. Today the top-scoring algorithm on the KITTI odometry benchmark is SOFT2, a second iteration by the same authors. Unfortunately, at the time of writing the publication of this work is still in review. SOFT-SLAM achieves impressing results, 0.88% of translational error and 0.0025 deg/m of rotational error, significantly better than ORB-SLAM2, while achieving lower average runtimes of 27ms versus

60ms.

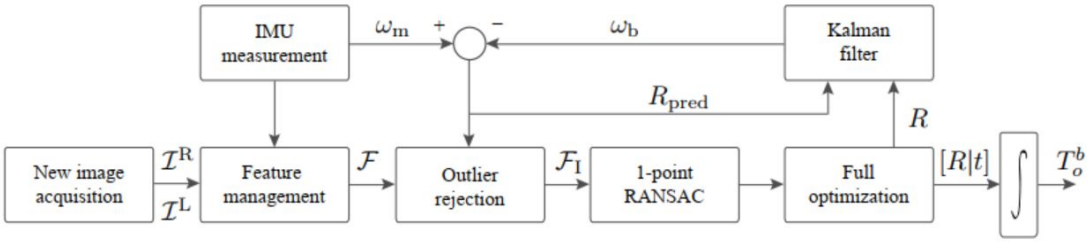


Figure 2.8: SOFT-SLAM architecture [25].

2.2.3 Visual Odometry: Deep learning methods

Drawing inspiration from the success of Convolutional Neural Networks (CNNs) in other areas of computer vision, DeepVO [26] was a cutting edge work in 2017. It tackles the problem of monocular visual odometry using deep recurrent CNNs in an end-to-end fashion. The proposed architecture needs to combine networks that learn geometric feature representations and ones that can derive connections between consecutive frames to resolve the VO problem and, as such, the authors combine a CNN and a Recurrent Neural Network (RNN) as exposed in Figure 2.9. The CNN is used for feature extraction: it takes the concatenation of two consecutive monocular RGB images as input, and extracts features whose representation is geometric and not associated with appearance or visual context because this system needs to be able to generalize the VO problem. Then, a deep RNN, is used to model dependencies between frames. Because RNNs are not suitable to directly work on a raw sequence of RGB images, the features that were extracted in the CNN are used as inputs. This network outputs a pose estimate for each time step, progressing over time with the movement of the camera sensor. Because it was one of the first works to integrate deep learning with visual odometry results on the KITTI benchmark are not up to par with other state-of-the-art approaches having a 5.96% translation error and a 0.0612 deg/m rotation error on average.

D3VO [27] is another approach to the same problem, by introducing a self-supervised neural network that predicts depth with the aid of DepthNet [28], pose with PoseNet [29] and uncertainty, it also calculates an affine brightness transformation to align the illumination of the training data. The first step involves illumination alignment, where the brightness transformation parameters are obtained from an affine transformation. For cases where this is not enough, for example the existence of non-Lambertian surfaces or moving objects that make the affine transformation difficult to model, another method is used, by calculating the photometric uncertainty. The next step is the main contribution of this work, integrating the prediction from DepthNet and PoseNet into a windowed sparse photometric bundle adjustment method much like in [17], where the total photometric error is minimized and D3VO is able to predict on areas with high reflectance like

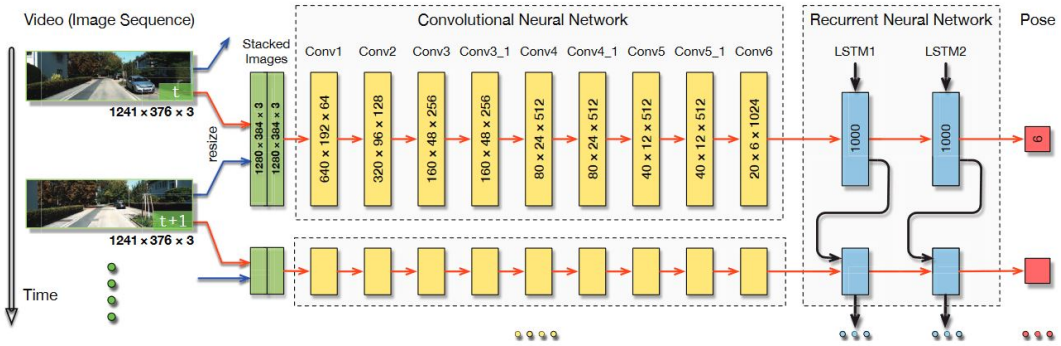


Figure 2.9: DeepVO architecture [26].

windows of vehicles or moving obstacles. Results show that this method achieves state-of-the-art performance in the area of monocular VO, getting comparable results to stereo/LiDAR systems that normally have better performance. D3VO presents much better results than the previous method with a 0.88% average translational error and a 0.0021 deg/m rotational error, these results are up to par with traditional VO techniques.

A more recent work developed by Zhu *et al.* and partially inspired in D3VO is DeepAVO [30]. It presents a four-branch network, to learn rotation and translation by focusing on different quadrants of optical flow input. The first part involves the calculation of the optical flow of consecutive RGB images, which works on the assumption of photometric consistency. Instead of using the well-known Lucas-Kanade method, the authors use a learning-based optical flow extractor, PWC-Net [16] that provides better results. Because pixel movement is more intense at the edge of the image and can be divided in 4 rough directions, one for each quadrant, the encoder module employs four parallel CNNs, as depicted in Figure 2.10, to exploit local visual cues. Then, the distilling module implements CBAM [31], which is a dual attention mechanism that calibrates the feature map in the channel and spatial domains. This mechanism focuses on objects that are closer to the camera (obvious motion), the objective of this module is to disregard redundant information and focus on the most important features that are necessary for odometry. Finally, a loss function considering both rotational and translational errors is designed. This work outperforms many learning-based and traditional VO methods. In terms of results DeepAVO has low runtimes averaging 12ms with the use of a GPU (NVIDIA Geforce Titan) and 53ms when only the CPU (Intel(R) Core(TM) i7-8700) is available, far lower than SOFT-SLAM that is considered to be a fast algorithm when traditional approaches are concerned. While having low estimation errors on certain scenes of the KITTI dataset, it doesn't score particularly well in the ones used to benchmark, achieving a 4.10% translation error and a 0.0125 deg/m rotation error.

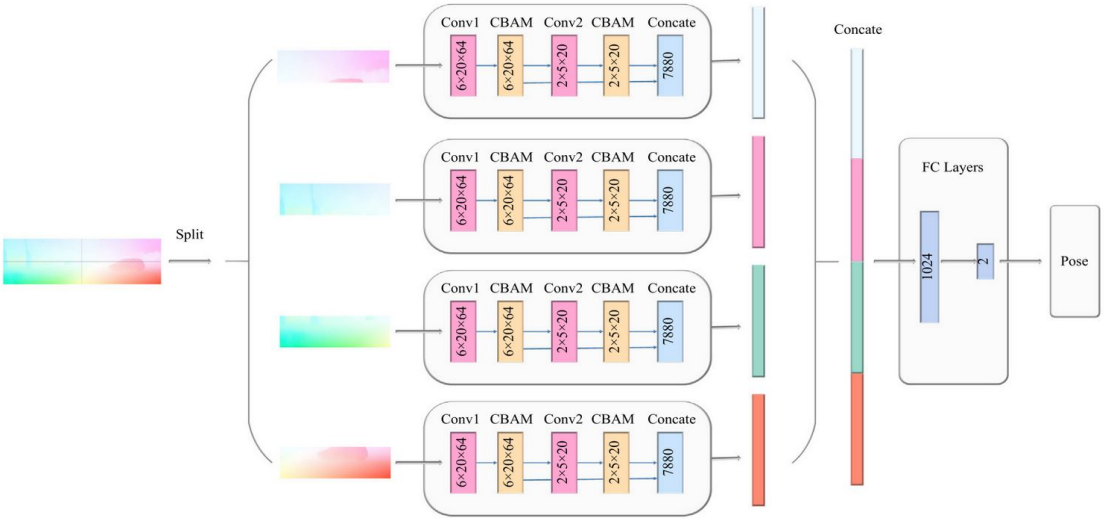


Figure 2.10: DeepAVO based monocular VO system architecture [30].

2.3 Point Cloud based Odometry

2.3.1 Point cloud based Odometry: Registration methods

Point correspondence methods were the first in the subject of laser odometry. Iterative Closest Point (ICP) [32], had its first versions in the 90s and can be considered as the most influential algorithm in this area. ICP is a method that finds the transformation between two point clouds and its most basic version has two steps: data association and transformation. The data association step aims to find correspondent points between two point clouds, which can be done using a nearest neighbor approach. The second step aims to minimize the distance between point pairs, by first computing the center of mass of each point cloud and aligning them, and then computing the rotation using Single Value Decomposition (SVD). This algorithm is run multiple iterations until a local minimum is found. Many authors build upon the foundations that ICP laid out, using it as a building block.

CT-ICP [33] is one of the best performing odometry systems on the KITTI autonomous driving benchmark. This work adapts the ICP algorithm to work in real-time, taking into consideration the movement of the laser sensor when scanning the environment, as it happens in an autonomous driving scenario. This is done by defining a continuous-time trajectory during the scan and discontinuous between scans. Each scan is composed of two different poses, one in the beginning and one in the end. Contrarily to many other methods, the beginning pose of a scan is not the same as the end pose of the previous scan, which helps to compensate for motion irregularities of the sensor. Despite this, the method still accumulates errors so the authors also propose a mapping method with loop closure, that when detected can be used to reduce error from pose to pose estimation. CT-ICP results on the KITTI odometry benchmark are quite good obtaining 0.59% translational error and 0.0014 deg/m of rotation error, having a run time of about 60ms to 65ms.

IMLS-SLAM [34] is a work from 2018 and it is also based on ICP. This work proposes a SLAM algorithm that only relies on 3D sensor data. Before applying the ICP based algorithm to estimate the pose, like the previous work, the movement of the sensor is compensated, and in an attempt to remove dynamic objects from the point cloud, as they can massively decrease the performance, a segmentation strategy is employed to identify and rule out small objects. This method diverges from classical ICP on the sampling strategy of the points: instead of using random points as samples, the various points are scored and listed according to their observability in terms of rotation and translation, typically points far from the sensor center and on planar zones. As this method uses scan-to-model matching, the model is defined as the previously localized LiDAR sweeps and uses the Implicit Moving Least Squares (IMLS) surface representation. IMLS-SLAM has slightly worse results when compared to CT-ICP, obtaining a 0.69% translation error and a 0.0018 deg/m rotation error. It is to note that this method is not implemented in real time in the KITTI dataset and it has a runtime o 1.25s per scan, because KITTI only provides a point cloud and not the raw LiDAR data, the authors mentioned that in the right conditions it could have a runtime of 50ms.

Another recent approach that relies on the basics of the Iterative Closest Point algorithm is MULLS (Multi-metric Linear Least Square) [35]. This work provides an efficient, low drift 3D SLAM system. This method firstly extracts geometric feature points and encodes them. Initially double thresholding is applied to extract the ground points that are further refined using RANSAC. The non-ground points are down sampled and fed into the principle components analysis module that applies a modified K-R nearest neighbor algorithm for the K nearest points in a sphere with radius R and extracts classified feature points. The next step applies multi-metric linear least square ICP: the inputs are a source and a target point cloud composed of the previously extracted multi-class feature points, then ICP is applied and the final estimation along with accuracy evaluation indexes are obtained. This ICP algorithm is slightly modified to increase the accuracy. This variation has 4 essential steps: multi-class closest point association, multi-metric transformation estimation, multi-strategy weighting function, and multi-index registration quality evaluation. This method proves to be efficient by being classified in sixteenth place in the KITTI odometry benchmark, having a 0.65% translation error and a 0.0019 deg/m rotation error. It is viable to employ MULLS on a moderate PC, it's run time varies between 80ms and 100 ms.

2.3.2 Point cloud based Odometry: Feature-based methods

LOAM [36], is one of the most influential works in the area, which works by corresponding feature points between scans to estimate the pose of the agent. The goal is to perform an ego-motion estimation with a point cloud perceived by a 3D LiDAR and build a map for the traversed environment. The algorithm extracts feature points on edges and planar surfaces and then matches them. First, feature points on edges and planar surfaces are selected and then sorted according to a c-value:

$$c = \frac{1}{|S| \cdot ||X^L_{(k,i)}||} || \sum_{j \in S, j \neq i} (X^L_{(k,i)} - X^L_{(k,j)}) || \tag{2.5}$$

Where S is a set of points and $X^L_{(k,i)}$ are the coordinates of point i from scan k in the coordinate frame L (LiDAR). Avoiding feature points whose surrounded points are selected or on local planar surfaces that are parallel to the laser beams. At the end of each sweep, the newly formed point cloud is re-projected and compared in real-time with the point cloud that is being formed and the goal is to find correspondents to the edge and planar features. After the correspondences of these are found, the distance between correspondent features from consecutive scans is calculated. Then the motion between two frames is calculated using the Levenberg-Marquardt method. Despite not being a very recent work LOAM holds up in the third place of the KITTI benchmark with 0.55% translation error and 0.0013 deg/m rotation error, runnig at a 10Hz rate.

Because LOAM is such a prominent work, several authors based themselves on it. This is the case for LeGOLOAM [37] that stands for Lightweight and Ground Optimized LOAM. This work was developed to be implemented in vehicles with lower computing power and no suspensions (this makes the LiDAR scans more distorted). The first step is to segment the point clouds, removing small clusters and preserving points that may represent big objects like tree trunks and road surfaces, saving them in a range image. Then features are extracted from these range images and classified as ground features or not. When it comes to the odometry algorithm, the matching of features across frames becomes faster because of the ground optimization procedure. The Levenberg-Marquardt method is then used to find the minimum-distance transformation between the two consecutive scans. As expected this method improves the efficiency of LOAM, achieving a 72% reduction in features used, having a run time of 6.1ms, when using an Intel Core i7 CPU while only processing the odometry module, if the mapping module is added this run time extends to about 100ms.

Some authors try to combine the strengths of both worlds and create methods that make use both of laser and visual odometry, such as LIMO [38]. This work aims to combine the highly accurate depth estimation from LiDAR and the powerful feature tracking capability of a monocular camera. The frames captured from the camera are used to extract features using the viso2 feature tracker and at the same time reject moving objects that reduce the accuracy of the pose estimations. The LiDAR data is used to estimate the scale from detected feature points, through several steps: selecting the neighborhood of the feature points, being aware not to choose only points in the same line to avoid singularities. The following step is foreground segmentation: if the feature points lie on flat surfaces there are no problems because the geometry around them can be estimated by the local plane. If these points are on edges there is the need to segment the foreground before the plane estimation, where only the closest points are taken into account. And last step is plane fitting during which, from the foreground segmented points, three are chosen that are able to have a big enough triangular area. Then fitting a plane to these points allows depth estimation. Frame to frame odometry is resolved through bundle adjustment, varying methods are used for urban and highway scenarios. This method achieves a mean translation error of 0.93% and a rotation error of 0.0026 deg/m, while running at 5Hz on quad-core 3.5Ghz CPU.

Similarly to LeGOLOAM, several other works try to improve on LOAM such as F-LOAM [39], that tries to reduce the computational burden by transforming the LOAM iterative processes

into a general solution, by creating a two-stage distortion compensation method instead of an iterative one. Other works such as R-LOAM [40] take this improvement in a different way, where the LOAM framework is combined with prior knowledge about a reference object in the environment to further improve the accuracy.

2.3.3 Point cloud based Odometry: Deep learning methods

There are some methods such as PWCLO-net [41] that solely rely on neural networks. This algorithm learns the LiDAR odometry from raw 3D point clouds in an end-to-end fashion with no need to pre-project the point cloud into 2D data such as range images. The inputs of the network are two point clouds, these are encoded by the siamese feature pyramid that extracts the hierarchical features of each point cloud. Then an attentive cost volume is used to associate the two point clouds and generate point embedding features that contain point correlation information. To obtain the pose transformation from these features an embedding mask is used with the added careful step of removing dynamic features, that may introduce errors in the initial pose estimation. Then an iterative pose refinement step is added to further improve the pose estimation. The PWCLO-Net architecture is shown in Figure 2.11. It has a translation error of 1.085% and rotation error of 0.0049 deg/m.

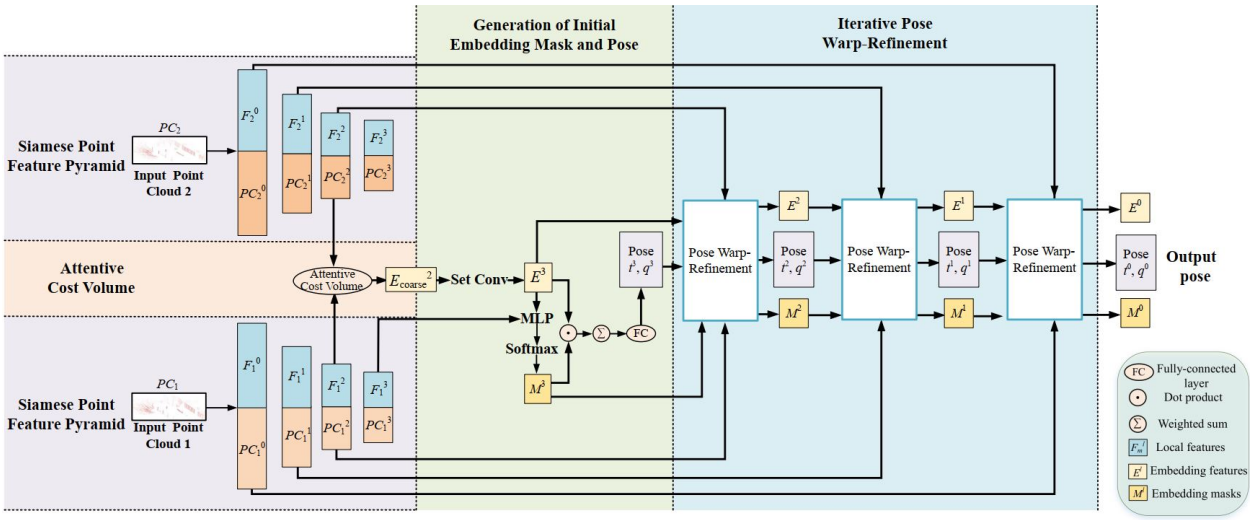


Figure 2.11: PWCLO-net archictecture [41].

Another recent approach leverages the power of deep convolutional networks to create a real-time LiDAR odometry estimation method. LO-Net [42] can learn feature representation for odometry estimation and mask dynamic objects in the environment, all in an end-to-end fashion, having three main modules: normal estimation, odometry regression, and mapping. As inputs, two consecutive LiDAR scans are accepted and processed by the first module that estimates a normal vector for each point taking its neighbors into consideration. The second module takes the two scans with the computed normals, calculates the features in each scan, and using an encoder-decoder architecture outputs a mask that compensates for dynamic objects. Then the end-layers

Table 2.1: Results of all available exposed methods on the KITTI odometry benchmark.

Method	Translation Error (%)	Rotation (deg/m)	Evaluated Scenes
DV-LOAM [19]	1.11	0.0050	00-10
ORB-SLAM 2 [22]	1.15	0.0027	11-21
SOFT-SLAM [25]	0.88	0.0025	11-21
DeepVO [26]	5.96	0.0612	03-10
D3VO [27]	0.88	0.0021	11-21
DeepAVO [30]	4.10	0.0125	11-21
CT-ICP [33]	0.59	0.0014	11-21
IMLS-SLAM [34]	0.69	0.0018	11-21
MULLS [35]	0.65	0.0019	11-21
LOAM [36]	0.55	0.0013	11-21
LIMO [38]	0.93	0.0026	11-21
PWCLO-net [41]	1.085	0.0049	07-10
LO-Net [42]	1.33	0.0069	00-10
LodoNet [43]	1.27	0.0066	00-10

of this module compute the 6-DoF pose between the two consecutive scans. To reduce the integration errors that are inherent to the pose estimation from scan to scan matches, the third module constructs a map and refines the pose estimation with a scan-to-map approach. This approach has a translation error of 1.33% and rotation error of 0.0069 deg/m. Average runtime is about 80ms using high a 1080ti GPU and an i7 quad-core CPU, both high power components.

LodoNet [43] is another recent work that relies on the already well-known deep learning methods applied in the area of visual odometry. The authors propose a method that transforms the LiDAR scanned point clouds into the image space. Then, feature extraction is done with SIFT and matched keypoint pairs are obtained between consecutive scans. These pairs are then fed into a convolutional neural network pipeline designed for LiDAR odometry. The translation and rotation errors on the KITTI benchmark are 1.27% and 0.0066 deg/m respectively.

Other recent and relevant works that employ the use of deep learning in the context of LiDAR-based laser odometry are PSF-LO [44] that uses parameterized semantic features and employs a dynamic and static object classifier, and CAE-LO [45] that uses unsupervised deep learning and utilizes compact 2D structured spherical ring projections. DMLO [46] is also worthy to mention, making feature matching applicable to LiDAR odometry by decomposing the pose estimation in two parts: a matching network that makes correspondences between two scans and rigid transformation estimation.

As a way to quickly compare all of the exposed odometry methods, Table 2.1 contains the results of all methods that were available on the KITTI odometry benchmark, and the published results by the corresponding authors in a compact format.

2.4 Dynamic object detection methods

As discussed earlier, knowledge-driven optical flow algorithms have been a topic of research for a long time in the area of computer vision, and in recent times have been surpassed in quantity by neural-based works, nonetheless, there is some recent work in this area. EpicFlow by Revaud *et al.* [47] is a knowledge-driven optical flow algorithm aimed to solve two of the biggest problems in optical flow: large displacement and occlusions, both very common in the context of autonomous driving. The first step involves taking two consecutive images and calculating a sparse set of matches between them, for this the authors decided to use DeepMatching [48], because it showed the best performance between all tested algorithms. The contours of the first image are also extracted using SED [49]. Given these two cues, a dense correspondence field between the two images is obtained by interpolating the matches. The obtained field is then used to initialize an energy minimization method that retrieves the optical flow. The proposed work surpassed other state-of-the-art coarse-to-fine methods obtaining an average end-point error (AEE) of 3.334% in the KITTI dataset and 3.686% in the MPI-Sintel dataset, despite this runtimes are around 16.4s which is not ideal for real-time applications. The pipeline of the algorithm is presented in Figure 2.12.

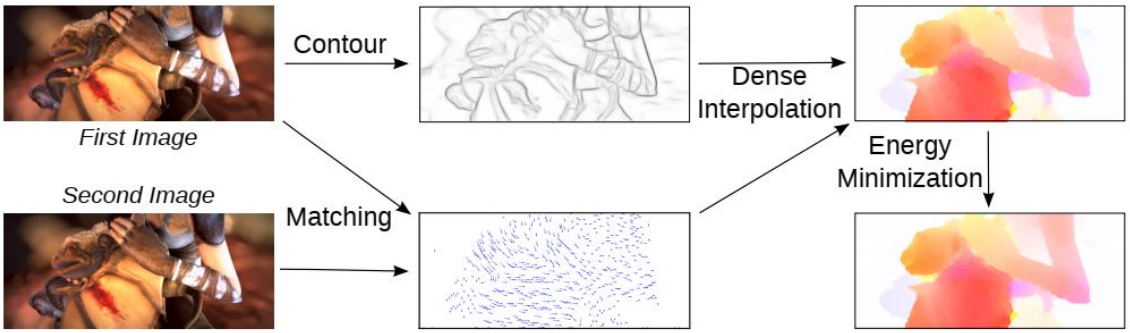


Figure 2.12: EpicFlow basic structure [47].

When concerning optical flow, data-driven approaches have surpassed the traditional ones in terms of accuracy and computing times, they also need to be addressed.

FlowNet [50] is considered a landmark on deep learning optical flow estimation. It was one of the first works to adapt the famous CNN model for the optical flow problem. It is able to predict the dense optical flow using two sequential RGB images. It uses a typical CNN architecture composed of a contracting part that takes two stacked images and an expanding part that outputs the optical flow results. This architecture is capable of estimating the flow with low inference times. Many further works used FlowNet as an inspiration for more complex architectures such is the case of FlowNet 2.0 [51] and PWC-net [16].

PWC-net [16] is a well-known CNN model for the detection of optical flow, using pyramid, warping, and cost volume. A general representation of the architecture is represented in Figure 2.13. Taking two input images the feature pyramid extractor creates an L-level pyramid of feature

representations using convolutional filters, where the input images are at the bottom level. The warping layer then warps the features from the latest image at the *l*th level to the prior one using the flow from level *l+1*. The cost volume layer associates matching pixels from both images and creates a cost volume where the cost is defined as the correlation between the features of the warped and non-warped images. The optical flow estimator is essentially a multi-layer CNN and takes features from the first image and the cost volume as inputs, outputting the flow at the *l*th level of the pyramid. The context network is an optional layer being composed of a feed-forward CNN, it refines the optical flow estimation. This method obtained an average endpoint error of 5.04% on the MPI sintel benchmark, and 2.16% AEE a 9.6% F1-all error on KITTI.

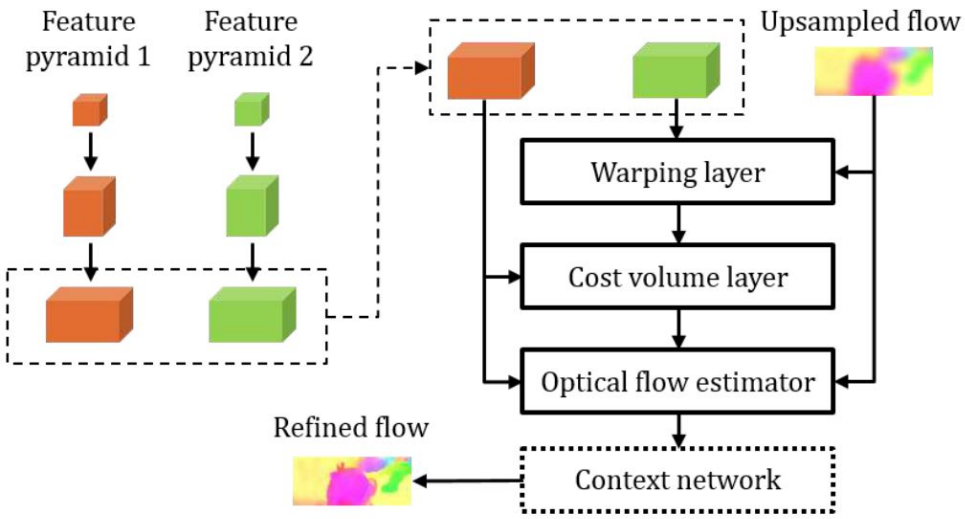


Figure 2.13: PWC-net pipeline [16].

RAFT [52] is recent work proposed by Teed and Deng that introduces a deep neural network for the detection of optical flow, where the objective is to estimate a dense displacement field when given two consecutive frames. It has 3 major components: a pixel-wise feature encoder, a correlation layer, and a recurrent GRU-based update operator. The first component employs a convolution network to detect features in both frames creating a low-resolution feature map. The correlation layer finds similarities between the feature maps of both frames and creates a four-dimensional correlational volume. The last module searches the correlational volumes and constantly updates the optical flow estimation. At the time of publishing, this work achieved a 5.10% of flow outliers (F1-all error), despite achieving better results this method is about twice as slow as PWC-net. An extension of this work was also proposed by the same authors, RAFT-3D [53]. This work is able to predict scene flow, which will be further discussed, using RGB-D images as inputs. The architecture is similar to RAFT but extended to work in the 3D domain and with ResNet50 [54] instead of a feature encoder to extract semantic information. The output consists of pixel-wise rigid body transformations.

In 2015 Menze and Geiger [55] presented a cornerstone work regarding object scene flow for

autonomous vehicles that served as a baseline to create the KITTI scene flow evaluation benchmark. Their contributions included a dataset for quantitative scene flow evaluation along with a model that estimates the flow of the scene. Relative to the model creation, the authors begin by assuming that the scene's 3D structure is approximated by a group of planar superpixels, and contains a limited number of dynamic objects. S is defined as the set of superpixels and O is the set of objects, each superpixel is associated with a region R_i and a random variable $s_i = (n_i, k_i)$ where n_i describes the 3D plane and k_i indicates the associated object, each object is linked with another random variable o_i that describes the rigid body motion. Note that each superpixel receives the motion of the associated object, with the 3D plane description, the 3D scene flow of each superpixel is fully determined. A cost reduction function is then defined to estimate the 3D motion:

$$E(s,o) = \sum_{i \in S} \varphi(s_i, o) + \sum_{i \sim j} \psi(s_i, s_j) \tag{2.6}$$

where $s = (s_i | i \in S)$, $o = (o_i | i \in O)$, $i \sim j$ means the set of adjacent superpixels, the first term in the function is designated as the data term and it models the assumption that the matching points across the four images (captured by stereo cameras) are similar. The second term of the function is the smoothness term, it encourages that neighbor superpixels have similar depth orientation and motion. The scene flow can be obtained by optimizing this equation. In terms of the novel dataset, the authors annotated 400 scenes from the KITTI dataset and defined an evaluation protocol to test several proposals for this problem. In terms of results, this method obtains about 10.63% of scene flow outliers, despite this, the runtime of this method is the best-case scenario is 120 seconds.

FlowNet3D [56] was designed with the intent to learn scene flow directly from point clouds in an end-to-end fashion. This network simultaneously learns deep hierarchical features of point clouds and flow embeddings that represent point motion. Adding to this, the proposed FlowNet3D architecture was applied on real LiDAR scans from KITTI and achieved greatly improved results in 3D scene flow estimation compared with traditional methods. The irregular structure in point clouds (no regular grids as in images) presents new challenges and opportunities for the design of novel architectures, which is the focus of this work. It consists of 3 layers. The first layer uses the PointNet3D++ [57] architecture to detect features in point clouds. This specific network was chosen because, in a point cloud, the set of points is irregular and orderless, demanding a translational invariant network and not other ones that apply traditional convolutions. The second layer (point mixture with flow embedding layer) learns geometric relations between two consecutive point clouds to infer motion, obtaining flow embeddings. In the third and final layer, these embeddings are upsampled and the refined scene flow is obtained. FlowNet3D achieves significantly better results than the previous method obtaining a 5.61% of scene flow outliers with an average end-point-error of 0.122m on the KITTI dataset.

PointPWC-Net [58] is a more recent approach but tackles the same problem as the previous work. This method builds upon the already explained PWC-Net used for optical flow by adding a novel way to compute the cost volume of 3D point clouds which can be quite complicated since these are typically unordered and their density is not uniform like in an image. When given features

from two point clouds, the matching cost is obtained using the MLP universal approximator to obtain the relation between two points:

$$Cost = MLP(concat(f_i, g_j, q_j - p_i)) \tag{2.7}$$

where *concat* means concatenation, f_i is the feature from a certain point p_i from point cloud P and g_j is the feature from point q_j from point cloud Q. After the cost is obtained normal convolutions cannot be applied, therefore the authors propose to aggregate the costs in a patch-to-patch way. When the cost volume layer is built the rest of the process is then similar to the already covered PWC-Net. This method achieves a 3D end-point error of 0.043m and an outlier ratio of 20.72%.

As explained before, estimating the motion of external objects is especially difficult when the sensor is moving, some works that incorporate scene flow estimation and dynamic object detection into odometry algorithms will be exposed.

The work presented in 2021 by Chen *et al.* [59] aims to address the problem of segmenting moving objects from 3D LiDAR scans in the context of autonomous driving. To achieve these goals the system needs to be accurate and fast, to enable the autonomous vehicle to make decisions on time. The first step involves converting the point cloud into a range image. This conversion consists of mapping the original point (x,y,z) into spherical coordinates and then transforming these into image coordinates. Each image coordinate will be associated with a 3D point, which makes it possible to associate several features with each pixel such as range, x y z coordinates, and remission. The second step involves creating residual images, through a three-step procedure: compensating the ego-motion by transforming previous scans into the local coordinates, then past scans are re-projected into the current range images, and finally the residual is computed for each pixel. Although the perception of movement can be noticed in the residual images in Figure 2.14, these residuals alone are not enough to extract the movement of objects, so these images are concatenated with current range images. The last step involves feeding these fused images into CNNs. The authors did not develop new networks but tested existing ones such as SalsaNext [60], RangeNet++ [61], and MINet [62]. All of these networks are state-of-the-art semantic segmentation networks, which take LiDAR range projection images as inputs and can achieve real-time operation and run faster than the frame rate of the LiDAR sensor. As previously said the fused images are fed into the networks that are retrained and evaluated for performance. Ablation studies were performed to conclude which of the segmentation networks would be the better choice, SalsaNext outperformed the other alternatives in every scenario, while also achieving state-of-the-art performance in the area of moving object segmentation (MOS). Further tests were made where the MOS algorithm was combined with an odometry one, and results prove that adding it made a positive difference. By adding the MOS module to SuMA [63] the translational error improved from 1.39% to 0.99% and the rotation error decreased from 0.0034 deg/m to 0.0033 deg/m.

Other recent advancements in the area were exposed by Pfreundschuh *et al.* [64], whose work leverages the power of deep learning to detect dynamic objects and can be divided into two main parts: offline and online. In the offline stage, an occupancy grid-based pipeline is used to detect

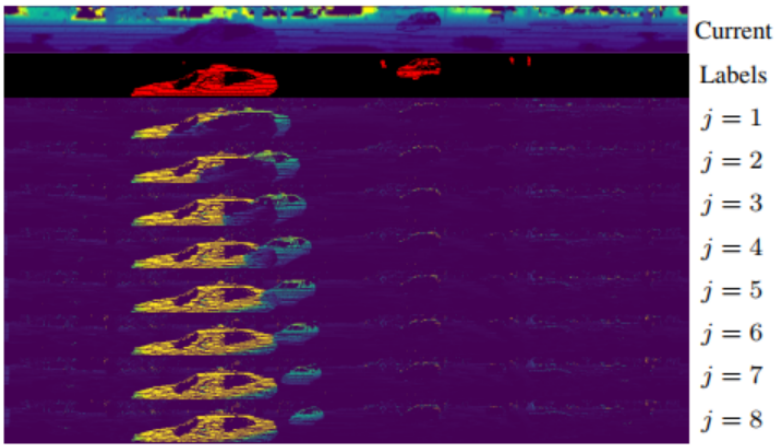


Figure 2.14: Residual images generated between the current frame and the last j-th frame[59].

dynamic objects in 3D LiDAR point clouds and automatically generate labeled data sets that are used to train a neural network. This occupancy grid is created in several steps, this process takes all the points scanned in a LiDAR revolution as input. Then, in the first step, ray-tracing is performed to classify voxels as free or occupied. If in two subsequent scans the state of a voxel changes from free to occupied it is classified as a candidate for moving. To avoid erroneous detections in the case of slow-moving objects, more than a singular past frame is used. After this, ground objects are removed and a clustering algorithm is applied to cluster points from the same objects and get the final labeled data. Because this process is computationally demanding and cannot be applied in a real-time application, the labeled data is used to train a neural network, in this case, 3D-MiniNet [65]. In the online part of this work, the already trained 3D-MiniNet takes the scanned LiDAR point clouds and detects the moving obstacles in real-time. The authors integrated this system in the well-known LOAM algorithm, by using their method to classify each detected feature as static or dynamic and therefore removing the dynamic features from the process. Their version of LOAM outperformed the classic one by obtaining 39.6 % less translational drift. The full pipeline is represented in Figure 2.15.

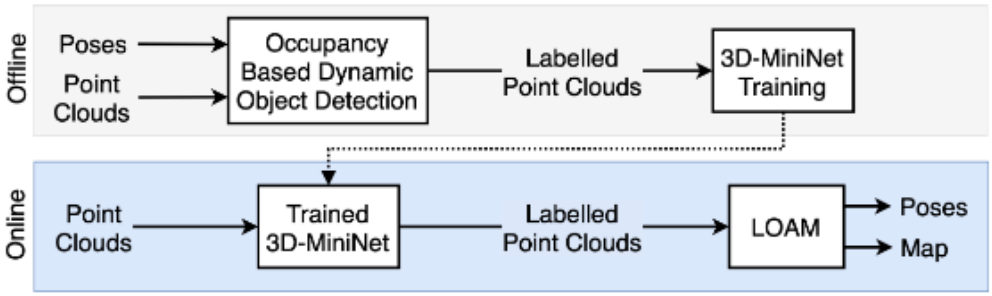


Figure 2.15: Full pipeline by Pfreundschuh *et al.* [64].

A different type of approach to detect dynamic objects makes use of a map. As it was already mentioned, many ego-motion estimation algorithms also build maps simultaneously as the autonomous agent traverses the environment, these are the SLAM algorithms. SuMa++ [66] is a relevant and recent work by Chen *et al.* that expands their previous work SuMa [63], that estimates a robot position by analyzing changes in a surfel based map, to also detect and remove dynamic objects, and improve the pose estimation. Each scanned LiDAR frame is converted into a 2D projection. Then, each frame is segmented by RangeNet++ [61] and each point is attributed a semantic label, after this, to finally complete the map creation process the image is converted back into a 3D projection to generate or update the world map. To detect the moving objects, semantic label consistency is checked between a new observation and the world model (surfel-based map), if the labels are inconsistent, it is assumed that those points belong to a moving object. A penalty is added to the stability term in the Bayes algorithm and after several observations, the surfels corresponding to the moving objects can be removed from the map and the ego-motion estimation calculation. As expected this method improves the original method: the rotational error improves from 0.0036 deg/m to 0.0029 deg/m and the translational decreased from 0.83% to 0.7% in the KITTI odometry training dataset. Similar to this, the work of Pagag *et al.* [67] is also of worthy mention, where the goal is to remove dynamic objects from point cloud maps. In short, the authors build an occupancy map by using an object classifier [68] where voxels represent the occupancy state over an extended period. This occupancy map is used as a filter to remove dynamic objects from scans before adding them to the map.

Although more and more common, not all recent approaches leverage the power of neural networks for dynamic object detection. Yoon *et al.* [69] present a mapless online detection of dynamic objects in 3D LiDAR, that differs from the already mentioned approaches by not using any type of artificial intelligence or neural networks. This method is model-free and does not require any type of map. Calculating the odometry of the moving sensor platform based on previous work [70] is the first step. The ego-motion estimation algorithm works with key points which need to be selected, to chose them normalized intensity is defined as Ir^2 (based on Lambertian reflectance) where I is the point intensity and r is the point range, and points are selected based on the eigenvalues of the covariance matrix of their k-nearest neighbors. Point matching is similar to the already explained ICP algorithm. Odometry estimation is solved as Gauss process regression. After trajectory estimation, a point cloud comparison is computed between two sequential scans, that are captured on different time intervals, making moving objects cause discrepancies that are calculated using error metrics, which are high and low for dynamic and static objects respectively. After this procedure, as many as half of the points are labeled as dynamic incorrectly, so refinement is carried out. Remembering that a LiDAR continuously sweeps lasers, the laser ray path from the sensor to the endpoint is defined as free space. Given a possibly dynamic point and a reference scan, it is necessary to know if this point is inside or outside of the free space, because of their movement only dynamic points cause free space discrepancies, by being inside the free spaces of other scans, therefore using this method the labels of the dynamic points are refined. The scans are then arranged into binary images where one represents the dynamic points and zero

the static, a sliding box filter is applied to these images to further rule out mislabeling due to finite LiDAR resolution. In a final step, the dynamic points are clustered and a region growth algorithm is applied.

Pinto et al. [71] present a flow-based motion perception technique for a robotic surveillance moving system using a monocular camera and limited computing resources. This system uses an innovative dense optical flow technique, HybridTree, to perform surveillance by motion detection while the sensor is positioned on a moving robot. This optical flow technique has two distinct phases: expectation and sensing. In expectation, an image is analyzed by looking for regions and cues that could retain different characteristics or motions. The idea is that the expectation phase guides the sensing. Here a hybrid base technique combines local and global optical flow techniques (Lucas Kanade, Horn Schuck). These different methods are combined using the cognitive information from the first phase to obtain an accurate and low-power demanding optical flow system. The same authors also presented another work [72] that proposes to cluster dense flow fields. This could be useful for detecting the moving objects present in a frame. The Wise Optical Flow Clustering technique is an evolution of the work presented in [73]. It has two phases: evaluating and resetting. Evaluation takes advantage of the polar representation of the optical flow and identifies regions of the field that retain different motion models, and the main objective is to guide the pixel clustering. In the resetting step, an algorithm based on the classic watershed method uses the previously gathered information as a starting step. This allows the segmentation of moving objects with low computing times. These ideas, while discussed in a different context, present a lot of similarities to the autonomous driving scenario: a moving sensor, real-time operation, and low computing power demands.

The work of Rath *et al.* [74] proposes an algorithm for the detection and tracking of moving objects (DATMO) for intended use in humanoid robots in GPS-denied environments. Although not being targeted to the application of autonomous driving, the principles are the same in both application scenarios. The main concept behind this work relies on observing the relative motion between the LiDAR sensor and the surrounding objects. As seen in Figure 2.16, the algorithm takes two inputs: a raw point cloud and the current LiDAR sensor pose, which are sampled independently and therefore must be synchronized. The second step involves removing the ground points from the point cloud because they make clustering segmentation a difficult task by creating lines that connect individual points. After ground point removal, euclidean clustering is applied, where points are clustered taking into consideration the euclidean distance between them, considering clustered points as individual objects. Then clusters/objects are matched across sequential frames, which only works if the LiDAR sensor has a high enough sampling rate, the sensor's velocity should be within a specific limited range and the spatial positions of clusters between two frames should not change dramatically. It is important to point out that these limitations may not be deal breakers in the case of the humanoid robot presented in this work but maybe an impediment in the context of autonomous driving. Moving object detection is done by applying the algorithm proposed by Kammerl *et al.* [75] that proposes the detection of two types of motion: movements of entire clusters with respect to the inertial frame and movement of a subset of points within a

cluster in relation to the cluster itself. The consistency of movement during several frames is then analyzed to remove false positives. For the removal of such points in the cloud that will be used for ego-motion estimation, the centroid of each cluster is computed and the individual points are removed using a k-nearest neighbor approach, therefore removing all dynamic objects from the point clouds. In an indoor office environment, this method presented a 91.42% accuracy.

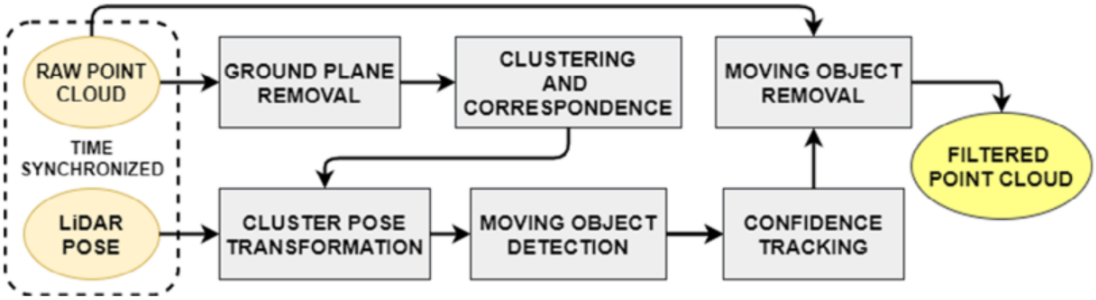


Figure 2.16: DATMO architecture [74].

Cluster VO [76] is a recent stereo visual odometry algorithm that simultaneously calculates the ego-motion and the movement of surrounding objects, organizing them into clusters. This algorithm takes synchronized and calibrated stereo images as inputs, and using the YOLO object detection network [77] for each frame, semantic bounding boxes are extracted and at the same time features are detected using the ORB detector [24]. With the aid of a map, these bounding boxes and features are matched with previously selected clusters and landmarks respectively, through a multi-level probabilistic association formulation. Then the landmarks observed in the current frames are clustered into different rigid bodies using a heterogeneous conditional random field, that works by minimizing an energy equation. Finally, the state estimation step optimizes all states over a sliding window. In terms of results, this method improves for example on ORB-SLAM2 specifically in dynamic environments.

Some methods employ the fusion of RGB images and LiDAR scanned point clouds to estimate moving objects in the environment. This is exactly what Rashed *et al.* [78] propose for detecting moving objects in low light scenarios in the context of autonomous driving. The objective is to complement optical flow algorithms that are inefficient in low light conditions with the presence of LiDAR data that is illumination independent. To interpret the motion from the LiDAR data, the work of Vaquero *et al.* [79] that models optical flow maps that are referred to as "lidarFlow" is used. The optical flow from the camera-captured images is generated using Flownet2 [51] and referred to as "rgbFlow". The authors propose a three-stream mid-fusion network with three encoders for the RGB images, "rgbFlow" and "lidarFlow", that can be observed in Figure 2.17. This architecture successfully implements a 4.25% relative improvement on the KITTI benchmark and 10.1% improvement on DarkKITTI (a low light only variation of KITTI). In normal lighting conditions, this method presented a 51.46 IoU for moving objects and in dark conditions a 43.5 IoU.

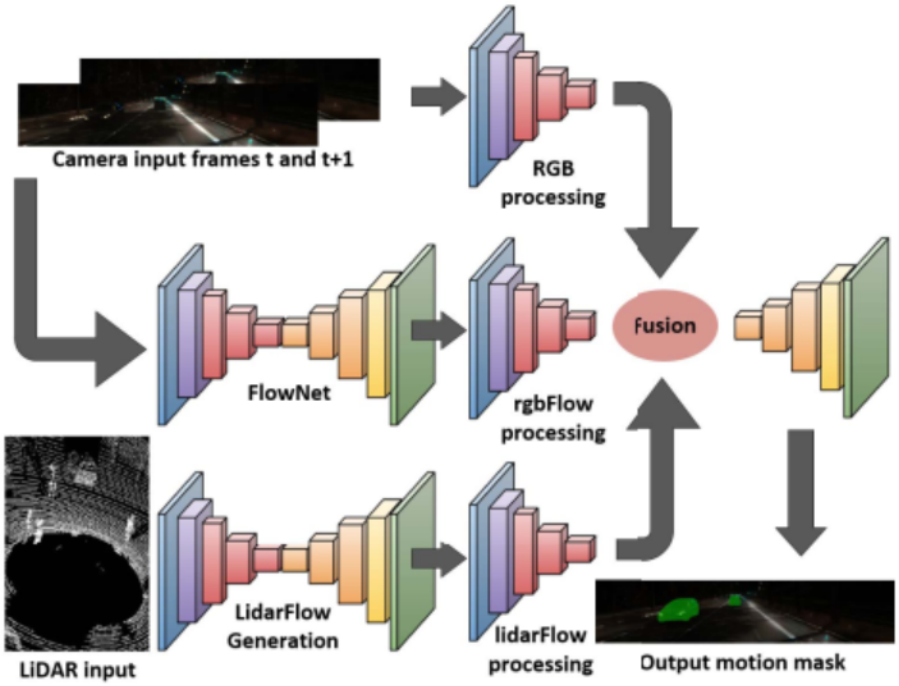


Figure 2.17: FuseMODNet architecture [78].

2.5 Critical review

Regarding the literature review, in VO traditional methods tend to generally outperform the deep-learning methods, despite normally being more computationally demanding and having longer runtimes. The two VO approaches that stand out are SOFT-SLAM and D3VO obtaining the best overall results. Despite this, by checking Table 2.1, it is simple to conclude that LiDAR-based odometry algorithms generally outperform the visual ones, again the traditional methods tend to achieve better results consistently. Despite being an older method, LOAM obtains the best overall results in terms of translation and rotation errors. Deep learning approaches tend to have worse results. When comparing visual odometry with point cloud-based odometry, the latter tend to perform better overall, despite this it can be more computationally demanding because of the unordered nature of point clouds. These methods also suffer from the limitations that were already discussed such as adverse weather conditions where the sensors, these being cameras or LiDARs, struggle to acquire noise-free representations of the environment. The presence of dynamic objects also affects the ego-motion estimation capabilities of these algorithms, making it necessary to be able to find a way to detect the dynamic agents and remove their influence from the estimation.

In terms of detecting dynamic objects, different types of approaches were explored regarding both the two-dimensional and three-dimensional spaces. Regarding optical and scene flow, deep learning approaches have caught up to traditional methods, DL constitutes the state-of-the-art in terms of performance and runtimes. When analyzing the different approaches that implement

dynamic object detection into their odometry or mapping algorithm, it is safe to conclude that this feature is valuable and confers improvements in all presented works. The best methods normally use some type of deep learning module to infer the movement of objects, and in all cases warping images or point clouds proved critical to detect dynamic objects from a moving vehicle.

To remove the influence of dynamic objects from the ego-motion estimation process there are two basic steps: detect these moving points or pixels using the already covered techniques like optical flow and segment the dynamic objects in the scene. This segmentation can be a complicated procedure that current algorithms struggle to do efficiently. It is also necessary to integrate the detection and segmentation of dynamic objects into an odometry algorithm in such a way that it improves the estimation results without substantially increasing the computational load.

Chapter 3

Dynamics-Aware Visual Odometry System

Visual odometry systems are becoming increasingly more relevant as the current state of autonomous vehicles advances. Manufacturers are racing to fit their products with autonomous capabilities. In this way, the ability to estimate the vehicle's trajectory robustly is essential. Visual odometry, specifically in the context of autonomous driving, consists of estimating the vehicle's current position and past trajectory. As a camera continuously records the movement of the car, it is possible to infer the frame-to-frame 3D movement estimation. The ego vehicle's odometry is inferred by extracting the movement from the background. If a moving object is used instead of the background, it will give a false sense of movement, and the ego-motion estimation will not be accurate. The objective of this system involves estimating the ego-motion using a camera sensor. At the same time, the other moving objects in the scene should be detected and removed from the estimation procedure.

3.1 Introduction

This dissertation presents a hybrid visual odometry system that uses a combination of deep and knowledge-based approaches to estimate the ego-motion of a vehicle. Dynamic feature elements were integrated within the developed VO system to increase the robustness against dynamic objects in the scene.

The development and testing of this system were mainly performed using the KITTI [80] autonomous driving dataset as it is the largest and most used dataset in the area of autonomous driving, especially in the sub-field of odometry estimation. It provides accurate trajectory ground truths and a vast ammount of data, collected by different sensor types such as stereo cameras and LiDARs. Nonetheless, more data was used to train the deep learning optical flow estimation and segmentation networks, such as the FlyingThings3D [81], HD1K, and an extension of KITTI MoSeg [78] datasets, as large amounts of data are necessary to train such networks, to avoid overfitting.

The developed algorithm estimates the pose of the vehicle by integrating several frame-to-frame 3D velocity measurements without the use of loop-closure, while being robust to the presence of dynamic objects in the environment. Figure 3.1 depicts the general architecture of the dynamics-aware visual odometry system. This section covers the main aspects of the proposed system while explaining different sub-modules and interfaces. The system uses a single RGB stereo camera to gather the necessary information. The input of the system consists of sequential RGB frames taken from the left and right perspectives. The first step involves the estimation of the dense disparity between the left and the right views. For this purpose, the deep learning architecture PSM-Net [82] is a state-of-the-art method that performs accurate disparity estimation from two RGB frames. The optical flow estimator takes sequential frames from the left camera and estimates the dense optical flow field. Here, various deep learning approaches were used and compared, ranging from FlownetS [50], RAFT [52], and RAFT-3D [53]. The specifics of each network and the used training regiments will be discussed in more detail. As mentioned before, optical flow provides an excellent cue to infer the movement of objects in the scene, and serves as the base for calculating the sensor's ego-motion as well as detecting moving objects. The deep learning segmentation architecture U-Net [83] was adjusted to the dynamic object segmentation task, taking the optical flow and the disparity estimations as inputs. Then, the 3D velocity between frames can be estimated utilizing the optical flow and depth information. The dynamic object masks help to remove the interference caused by non ego-motion such as cars, trucks, or cyclists. The vehicle's pose over time can be recovered by integrating the frame-to-frame 3D velocity estimations. The following sections will expose the implementation details of this system and its different sub-modules.

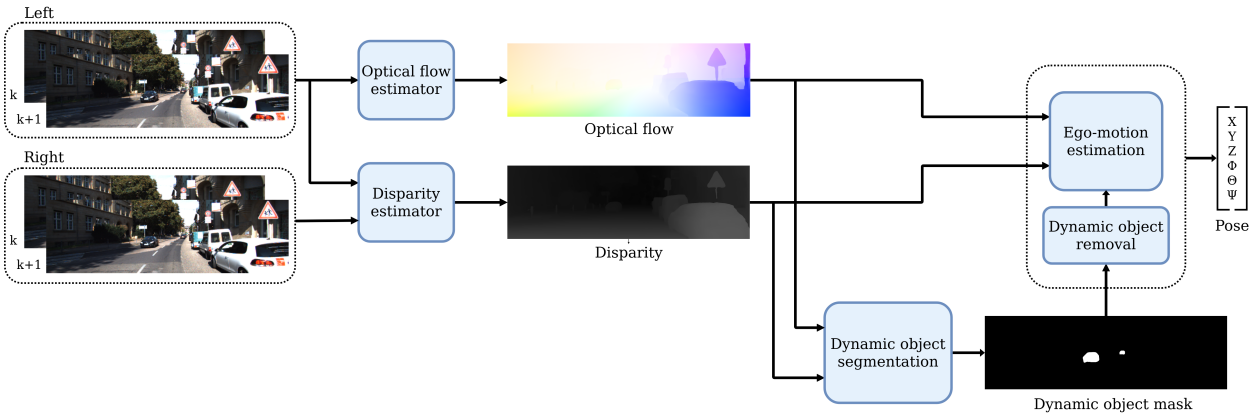


Figure 3.1: General architecture of Dynamics-Aware Visual Odometry System.

3.2 Disparity and depth estimation

As already mentioned, the first step of the proposed algorithm involves estimating the dense disparity using the left and right perspectives obtained from a stereo camera. Disparity is defined as

the difference in image location of a 3D point when captured by two different cameras. Using traditional methods, disparity can be recovered by finding correspondences along epipolar lines. However, recovering a dense estimation is computationally expensive, and requires a high degree of fine-tuning, that could vary from situation to situation. Recent deep learning architectures can do this job faster and more accurately, as is the case of PSM-Net [82], which is a pyramid spatial matching network composed of two modules. A spatial pyramid pooling module harvests features by concatenating representations from sub-regions with different sizes and a 3D CNN for cost volume regularization and disparity regression. PSM-Net is one of the top-scoring methods in the KITTI depth estimation benchmark and its code and trained models are openly available. The specific model used for this work was trained on a synthetic scene flow dataset with more than 35 000 images, and tuned on the KITTI training set for 300 epochs explicitly for the driving scenario. A disparity estimate is depicted in Figure 3.2.

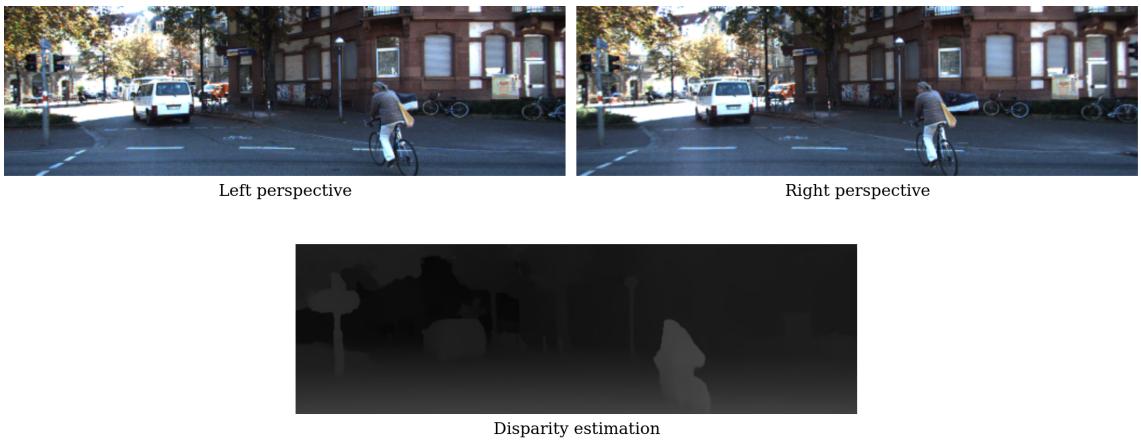


Figure 3.2: Example of a disparity estimate from PSM-Net in grayscale format; the left and right perspectives were used as inputs [80].

As disparity only informs about the difference in image position of two corresponding pixels taken by different cameras, it is necessary to use this information to compute the corresponding depth of each pixel. If one takes into account the front parallel stereo assumption that considers row aligned, undistorted and co-planar images with parallel optical axes, the depth of each pixel can be estimated using equation 3.1.

$$Z = \frac{fT}{x^L - x^R} \tag{3.1}$$

Where Z is the calculated depth, f is the camera's focal length, T is the baseline between the two centers of projection, and $(x^L - x^R)$ is the estimated disparity provided by PSM-Net, the focal length and baseline should be known *a priori*. With this method and the parameters of the stereo rig it is possible to estimate a depth for each corresponding pixel. The depth estimates are not exact as the precision is impacted by the baseline of the rig and the distance to the objects.

3.3 Optical flow estimation

Optical flow estimation was briefly addressed in Section 2.4. It was established that recent approaches that leverage the power of deep learning tend to obtain significantly better results while also averaging lower inference times, especially in the case of dense optical flow. As the quality of the optical flow is extremely important for the success of later stages of this algorithm, several approaches were studied and analyzed. Sequential images from the left perspective were used as inputs.

The first studied approach was FlownetS [50], one of the first CNNs used to estimate optical flow. The intent of using this specific architecture relied on its fast inference times, perfect for real-time operation, and its relatively simple encoder-decoder architecture. One channel was added to the network's input, converting the RGB input to RGB-D, using the disparity estimate from PSM-Net. Disparity can be considered a particular 1-D case of optical flow. This extra information could help to better differentiate moving objects for an easier segmentation in a later stage while also improving the overall quality of the optical flow. Generally different objects are at different depths, this assumption provides good cues for the optical flow estimator, making it easier to infer the flow of each object. To change the encoder-decoder architecture, the depth of the first three convolutional layers was augmented to consider the extra channel, and the network was entirely retrained, the original architecture is represented in Figure 3.3. It is challenging to find real-world optical flow datasets with accurate and dense optical flow ground truth; furthermore, these existing datasets are not large enough to train such networks. In a first stage, the network was trained using synthetic data, then, in second step, real-world data was used to fine tune the network. The used synthetic datasets were FlyingThings3D and MPI-Sintel while KITTI and HD1K are composed of real-world images.

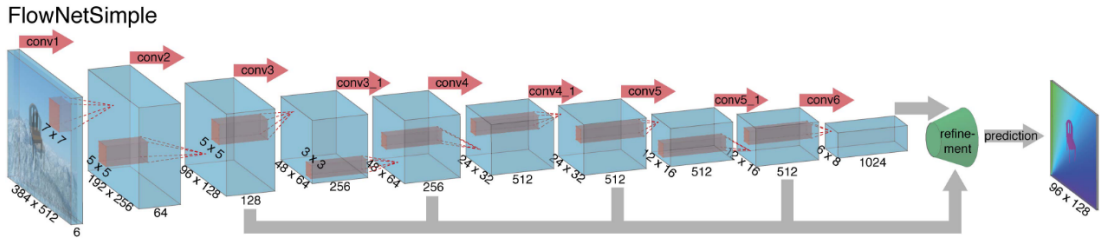


Figure 3.3: Example of FlownetS encoder-decoder architecture [50].

RAFT [52] was another optical flow network taken into consideration. It is more complex than FlownetS and with slighty higher inference times, of 0.10s, it presents significantly better results on the KITTI optical flow benchmark. This network also has the advantage of having a context encoder as it helps to better aggregate the motion of each specific object within its boundaries. The existence of this encoder could be useful in the moving object segmentation, as it extracts contextual image information and might help to differentiate between individual objects, i.e. two

different cars. As mentioned before the network achieves about 5.10% of flow outliers which is good for the ego-motion estimation as outliers drastically affect the performance of the algorithm.

Lastly, RAFT-3D [53], was also analyzed. This network is similar to RAFT, but it accepts RGB-D images as inputs, being able to compute the optical flow and the scene flow between two consecutive frames. This network is able to calculate pixel-wise rigid body transformations. To substitute RAFT's context encoder, a pre-trained ResNet50 [54] is used, it helps to group objects into different moving regions as the ResNet will add a degree of semantic information. From the covered networks, RAFT-3D obtains the best results on the KITTI optical flow benchmark. Nonetheless, this network has a large number of parameters, around 45 million, with inference times of 386 ms for 540x960 images, making real-time operation difficult. We tested this network to evaluate how the extra depth information also given by PSM-Net, could improve the optical flow and the pose estimation when compared to RAFT. Both RAFT and RAFT-3D were used to estimate the optical flow using pre-trained models for the KITTI dataset. The comparison and analysis of the optical flow obtained by these networks is done in Chapter 4.

3.4 Dynamic object segmentation

After obtaining the disparity and optical flow estimates, the dynamic objects were segmented. For this, the well-known U-Net architecture was applied. Although initially developed for specific use in biomedical imaging, this architecture is currently used in various application scenarios. This network's name is inspired by its shape, depicted in Figure 3.4. In a summarized way, the downward branch down-samples the input image through a series of convolution and max pooling operations. In contrast, the upward branch up-samples the information through transpose convolutions and a series of concatenated skip connections.

In this case, the network's input is a 3-channel image composed of the horizontal flow, the vertical flow, and the disparity images. Optical flow will provide the movement information, and the disparity will provide good cues towards the shape of the objects. It is also important to note that optical flow and disparity inputs were normalized within zero and one to ensure good operation. An RGB representation of the input along with the ground-truth for moving objects is presented in Figure 4.5. As it can be seen, the concatenation of optical flow and disparity provides good cues for the scene's movement.

The used dataset was an extension of KITTI MoSeg [78], an annotation of the moving objects of KITTI composed of 12919 images. Training schedules were performed with RAFT and FlowNetS+Depth optical flow estimates to provide a comparison. For the segmentation task U-Net was implemented in PyTorch. Using the Adam optimizer, the network tries to maximize the dice-score which is a metric used to evaluate the similarity of two samples, defined in Equation 3.2, where TP- true positive, FP- false positive and FN-false negative:

$$DSC = \frac{2TP}{2TP + FP + FN} \tag{3.2}$$

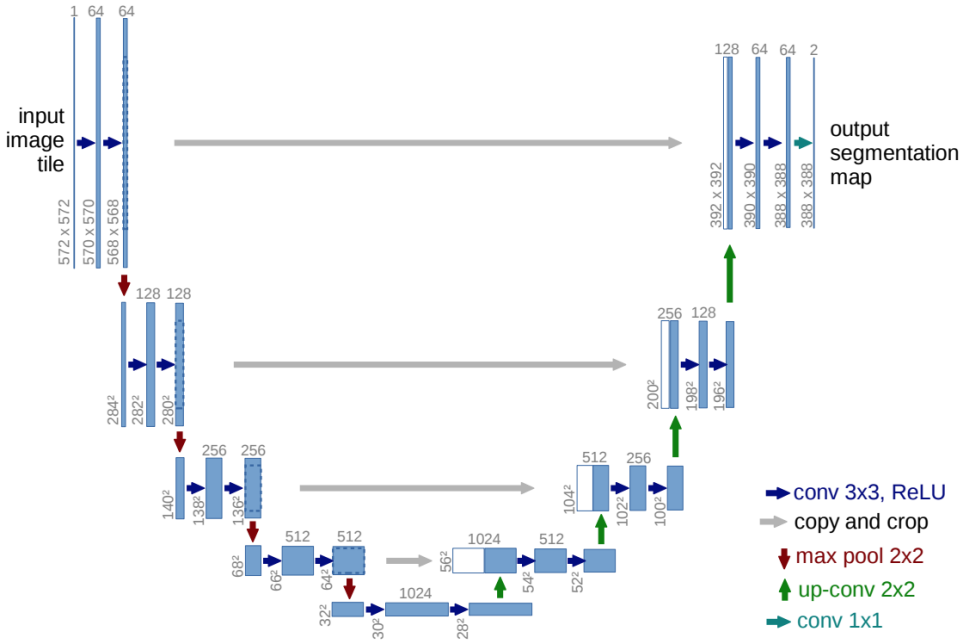


Figure 3.4: U-Net architecture [83].

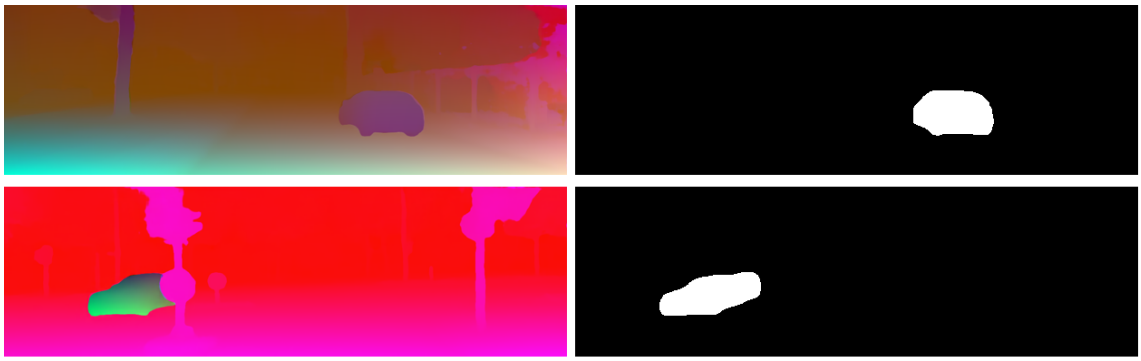
3.5 Odometry estimation

A motion field is the 2D representation of a 3D movement. For example, when a car equipped with a camera drives along a road, the camera records the perceived motion of the environment. As a point moves through a 3D path, its projection also moves according to a 2D path on the image plane. Figure 3.6 depicts this process in schematic form, assuming the perspective projection and a camera-centered coordinate frame. Considering all 3D points, which have a projection on the image plane, the conjunction of all corresponding 2D velocity vectors is defined as motion field. However, motion fields cannot be directly observed and differ from optical flow, which is the apparent motion of the brightness pattern. For example, assuming a smooth and perfectly uniform sphere illuminated by a stationary light source. If the sphere rotates the brightness pattern won't move, in this case the optical flow and motion fields are not correspondent. Despite this, optical flow is a reasonable estimate of the actual motion field in most cases. This basic assumption makes it possible to estimate the sensor's ego-motion, as explained in [84].

Assuming a static 3D point in space $P = [X,Y,Z]^T$, its movement is opposite to the camera movement and can be described by an angular velocity vector $\Omega = [\Omega_X, \Omega_Y, \Omega_Z]^T$ together with a translation velocity vector $T = [T_X, T_Y, T_Z]^T$:

$$\dot{P} = -T - \Omega \times P \tag{3.3}$$

Considering the perspective projection, the point P projects onto the image point $p(x,y)^T$ according to equations 3.4 and 3.5, where f (focal length) is the distance between the camera



(a) (b)

Figure 3.5: Example of inputs given to U-Net, (a) - 3 channel input image (concatenation): horizontal-flow, vertical-flow, disparity in RGB format; (b) - ground truth for moving objects.

coordinate center and the image plane along the Z axis.

$$x = \frac{fX}{Z} \tag{3.4}$$

$$y = \frac{fY}{Z} \tag{3.5}$$

As a 3D point moves, the corresponding 2D projection traces a path on the image plane, as shown in Figure 3.6 the derivative is considered the velocity of p, as depicted in Equation 3.6.

$$v \equiv (u,v)^T = \begin{bmatrix} \frac{\delta x}{\delta t} \\ \frac{\delta y}{\delta t} \end{bmatrix} \tag{3.6}$$

Substituting equations 3.4 and 3.5 into equation 3.6 generates:

$$v = \frac{f}{Z}\begin{bmatrix} \dot{X} - \frac{X\dot{Z}}{Z} \\ \dot{Y} - \frac{Y\dot{Z}}{Z} \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} f\dot{X} - \dot{Z}x \\ f\dot{Y} - \dot{Z}y \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} f & 0 & -x \\ 0 & f & -y \end{bmatrix}\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix} \tag{3.7}$$

Finally substituting equation 3.3 into equation 3.7 gives the 3D velocity:

$$v = \frac{1}{Z}\begin{bmatrix} -f & 0 & x \\ 0 & -f & y \end{bmatrix} T + \frac{1}{Z}\begin{bmatrix} -f & 0 & x \\ 0 & -f & y \end{bmatrix}\begin{bmatrix} 0 & 1 & -\frac{y}{f} \\ -1 & 0 & \frac{x}{f} \\ \frac{y}{f} & -\frac{x}{f} & 0 \end{bmatrix}\Omega \tag{3.8}$$

After some simplification, the 3D velocity can be represented as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{Z}\begin{bmatrix} -f & 0 & x \\ 0 & -f & y \end{bmatrix}\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} + \frac{1}{Z}\begin{bmatrix} \frac{xy}{f} & -f - \frac{x^2}{f} & y \\ f + \frac{y^2}{f} & -\frac{xy}{f} & -x \end{bmatrix}\begin{bmatrix} \Omega_x \\ \Omega_y \\ \Omega_z \end{bmatrix} \tag{3.9}$$

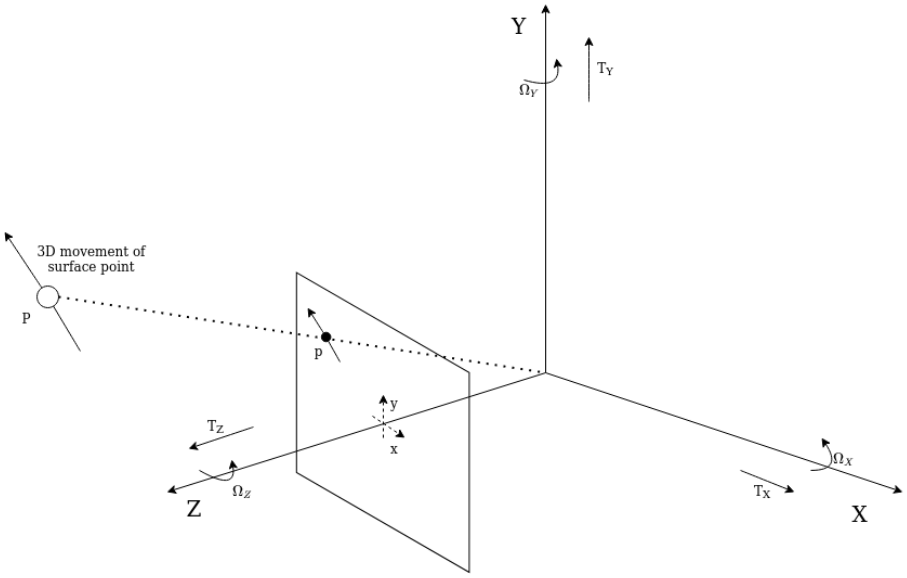


Figure 3.6: Image plane projection of a 3D point surface movement assuming the perspective projection and a camera-centered coordinate frame.

According to equation 3.9 it is possible to determine the 3D velocity of surface points if the depth and optical flow are known. It is also important to point out that equation 3.9 has six unknowns and only two equations, so it is necessary to have information about three optical flow points whose values are not co-linear. It is also possible to infer that only the translation is dependent on the depth information while rotation solely depends on the optical flow.

Given the optical flow estimated earlier and the depth derived from the disparity information, it is possible to calculate the frame-to-frame velocity estimation for the camera's movement. It is also easy to infer that the pose estimation will be highly dependent on the quality of the predictions, with poor optical flow estimates the results will be highly erroneous. Even if frame-to-frame predictions only have small errors, these will propagate over time, because each pose is dependent on all of the previous ones. This is normally perceived as drift, and is one of the most difficult challenges to surpass in the area of VO. Because both estimations are dense, one has a vast number of points. Therefore, the least-squares approach was used to robustly solve a problem with more equations than unknowns. For this purpose, the velocity estimation was formulated into the $Ax = b$ form such as:

$$\begin{bmatrix} -\frac{f}{Z} & 0 & \frac{x}{Z} & \frac{xy}{f} & -f-\frac{x^2}{f} & y \\ & & & \dots & & \\ 0 & -\frac{f}{Z} & \frac{y}{Z} & f+\frac{y^2}{f} & -\frac{xy}{f} & -x \\ & & & \dots & & \end{bmatrix} \begin{bmatrix} T_X \\ T_Y \\ T_Z \\ \Omega_X \\ \Omega_Y \\ \Omega_Z \end{bmatrix} = \begin{bmatrix} u \\ \dots \\ v \\ \dots \end{bmatrix} \tag{3.10}$$

Despite using the least-squares approach, the process is not accurate enough to estimate the ego-motion of the vehicle, because of the presence of large amounts of noise, especially in the optical flow. Removing as many outlier points as possible is necessary to improve the optimization process. In order to smooth and remove extreme outliers, a five-by-five median filter was applied to the optical flow images as this filter is robust to high local variations.

As mentioned before, it is also important to divide the images into static and moving objects, which is extremely important as using flow vectors that correspond with such objects will diminish the quality of the results. To estimate the movement of the sensor's platform, only the static points should be taken into account. For this, the already computed segmentation mask is applied, removing the pixels that belong to moving objects so that the least-squares optimization only takes the static points into account.

After removing the dynamic points, some outliers remain present in the current set , even with smoothing and processing, this happens for a variety of reasons. Image zones that contain, for example, the sky or far away zones contribute to non-accurate velocity estimates as the calculated depth might be erroneous because the maximum estimated depth depends on the stereo baseline. Furthermore, the depth estimation error varies with the squared distance as portrayed in equation 3.11, where ∂Z is the depth error, and ∂d is the disparity error. So it is necessary to remove the projections of points in the horizon and in the sky.

$$\partial Z = -\frac{Z^2}{fT}\partial d \tag{3.11}$$

The RANSAC algorithm was used to exclude outliers from the selected points. This iterative approach aims to separate a set of points into a group of outliers and a group of inliers. In a general way, RANSAC has three simple steps: the first step involves selecting a random set of data points and treating them as inliers. The second step uses the sampled points and computes the model parameters. The model is scored according to its number of inliers in the last step, given a specific threshold. These steps are repeated, and the best model after all iterations is used to determine which points are inliers and outliers. In this specific case, in each iteration, three random optical flow points are selected, and according to equation 3.9 the 3D velocity estimate is computed. Finally, inliers and outliers are selected to score the model and divide the image points into outliers and inliers.

One critical question focuses on the number of iterations performed to achieve the best possible results. The number of iterations, T, to succeed with a probability of P and an outlier ratio of e when sampling s points is given by equation 4.1.

$$T = \frac{log(1-P)}{log(1-(1-e)^s)} \tag{3.12}$$

For real-time operation it is necessary to minimize the number of iterations. Given that the used model has three parameters, the only way to significantly impact number of iterations, is to use high quality optical flow and disparity estimates, with low outlier ratios, this is why this work has a large focus on inspecting and evaluating the quality of optical flow estimates.

After selecting only the inlier points, the least-squares approach can be used, considering only the inliers. Given an optical flow and a disparity image, the calculated result will be a 3D velocity vector, $[T_x, T_y, T_z, \Omega_x, \Omega_y, \Omega_z]^T$. To estimate the sensor's pose over time, the prediction could be converted to a rigid body transformation that can be multiplied by the sensor's pose to obtain a prediction over time, pose n represented on the 0^{th} frame is given by:

$$T_{0n} = T_{01} \times T_{12} \times T_{23} \times ...T_{n-2,n-1} \times T_{n-1,n} \tag{3.13}$$

Conclusion

So far, the implementation of the Dynamics-Aware Visual Odometry System has been explained in detail. In a first step, deep learning architectures estimate the disparity and optical flow using sequential frames from a stereo RGB camera. These estimates are then used to create a moving object segmentation with the aid of U-Net and estimate the sensor's ego-motion using a knowledge-based approach. The various results regarding optical flow estimation, the moving object segmentation, and the ego-motion estimation will be exposed and analyzed in the following section.

Chapter 4

Results

4.1 Introduction

In this section, testing procedures will evaluate the performance of the developed VO system. As the presented method comprises several building modules, each block will be submitted to an individual testing procedure in the first stage. Then in a second step, the dynamics-aware visual odometry system will be evaluated in terms of ego-motion estimation. The used GPU is a Nvidia A100 tensor core GPU with 40GB memory. The optical flow estimators' end-point error will be evaluated in an extensive set of images. The results will be visually inspected in static and dynamic scenarios. To test the dynamic segmentation network, various environments where the objects have multiple motions and the environmental conditions, such as variable brightness, will be analyzed. The metrics used for this evaluation are the dice-score and the Intersection over Union coefficient (IoU).

Finally, the odometry system will be evaluated in static and dynamic scenarios. The effect of the dynamic mask will be quantitatively assessed. The method will also be analyzed against an existing stereo VO method to have a comparison against the state-of-the-art.

4.2 Optical flow evaluation

Several deep learning architectures were employed and modified to calculate optical flow estimates. The optical flow results have two primary purposes, to be used in the moving object segmentation step and to calculate the vehicle's ego-motion. To segment the moving objects, the optical flow estimates should present clear boundaries between moving objects and the static background. In fact, these distinct classes of objects should have significantly different optical flow movements, and each moving object should present clear boundaries, even if the estimates are not extremely accurate. The objective is different for ego-motion estimation. The average error for all flow vectors must be as minimal as possible, to have accurate odometry estimates.

Four architectures, presented in Table 4.1, were tested to determine which optical flow estimator would be optimal for each step. These were analyzed both quantitatively and qualitatively:

Table 4.1: Used datasets to train each optical flow network.

Method	Synthetic data			Real-world data	
	FlyingChairs	FlyngThings 3D	MPI-Sintel	HD1K	KITTI 2015
FlowNetS [50]	⋆		⋆		
FlowNetS+ Depth		⋆	⋆	⋆	⋆
RAFT [52]	⋆	⋆			⋆
RAFT-3D [53]		⋆			⋆

inspecting the average optical flow end-point error and visually examining each architecture's optical flow estimates. The end-point error (EPE) is the norm of the difference between the computed optical flow (v_{est}) and the ground truth (v_{gt}) as:

$$EPE = \| v_{est} - v_{gt} \|. \tag{4.1}$$

The evaluated architectures were: Flownet Simple [50], Flownet Simple with Depth added as an extra channel, RAFT [52], and RAFT-3D [53]. It is worthy to point out that the driving scenario presents specific challenges for optical flow estimation. Despite the presence of many different moving objects and the motion of the sensor itself, every object can generally be described as rigid. It is satisfactory to estimate one general motion for each moving object, as all points of the respective objects have the same general movement.

An existing model of FlownetS trained on the FlyingChairs and MPI-Sintel datasets was utilized. Regarding the altered FlownetS with the extra depth channel, this model was trained using the synthetic datasets FlyingThings dataset and MPI-Sintel, with a special emphasis on a driving subset of FlyingThings. The real-world HD1K and KITTI 2015 optical flow datasets, while not optimal, are the best real-world datasets with a focus on the driving scenario. They are not perfect because the HD1K only has gray images and offers sparse ground truth, as not all pixels have ground truth available. The KITTI dataset has the same problem of a sparse ground truth and is composed of a small set of images, about four-hundred. A pre-trained model on FlyingChairs, FlyingThings, and fine tunned using KITTI 2015 was used for RAFT. Finally, the pre-trained model used for RAFT-3D was trained on FlyingThings3D and fine-tuned on KITTI 2015. The different used datasets can be checked in Table 4.1.

Evaluating different estimators implies selecting a dataset that was not included in any training regiments. It would be optimal to evaluate them in real-world data, specifically in a driving scenario. The older KITTI 2012 was used for this purpose. Nonetheless, this dataset mainly consists of scenarios composed by static objects. Sequences with a high number of dynamic objects were selected from the raw KITTI dataset, to analyze the performance of the optical flow estimators in dynamic environments. For each architecture, the average end-point error was calculated for a pair of sequential images. Then, the result of all 197 sequential pairs was averaged to obtain a robust metric of the method's performance in a real-world driving scenario. The results are presented in Table 4.2.

By directly inspecting Table 4.2, RAFT and RAFT-3D are clearly superior, given the networks

Table 4.2: Average EPE across the KITTI 2012 optical flow dataset.

Method	Average end-point error
FlowNetS	0.026057
FlowNetS + Depth	0.055638
RAFT	0.006984
RAFT-3D	0.008928

complexity. The added depth does not seem to improve the EPE. However, one must have in mind that the evaluated dataset is mainly composed of static scenes where the added disparity may not improve the estimation, such as the ones presented in Figure 4.1. In these examples, there are no considerable depth variations and no different objects at different depths related to the camera sensor.

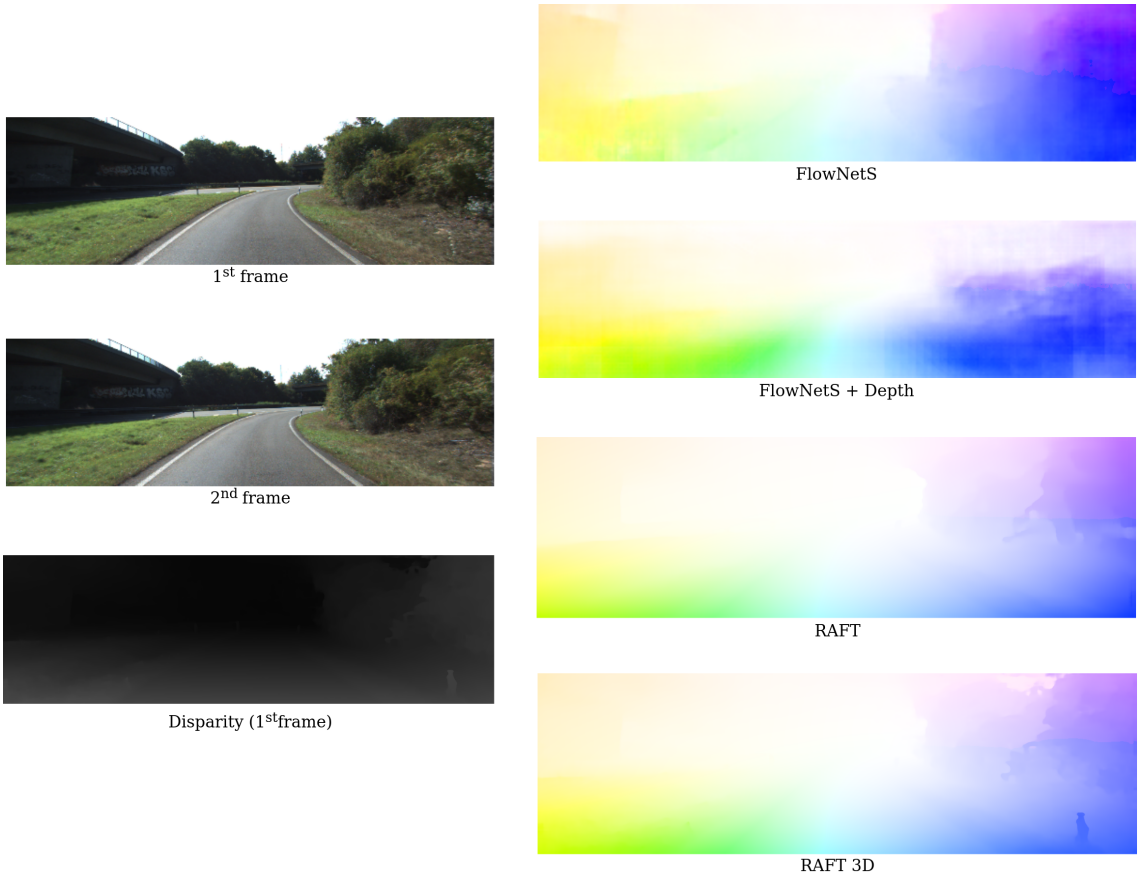


Figure 4.1: Optical flow estimates given by different architectures. These are computed between the first and second frames. In this case, the disparity image doesn't carry substantial information to aid FlowNetS + Depth or RAFT-3D.

Nonetheless, it would also be interesting to analyze the different flow estimators in populated environments with static and dynamic objects. Here the added depth, as well as the context encoder in RAFT and the ResNet50 in RAFT-3D, contribute with contextual and semantic information for

the optical flow estimation. As KITTI 2012 does not have such scenarios, the raw KITTI dataset was used, and the estimates can be seen in Figure 4.2.

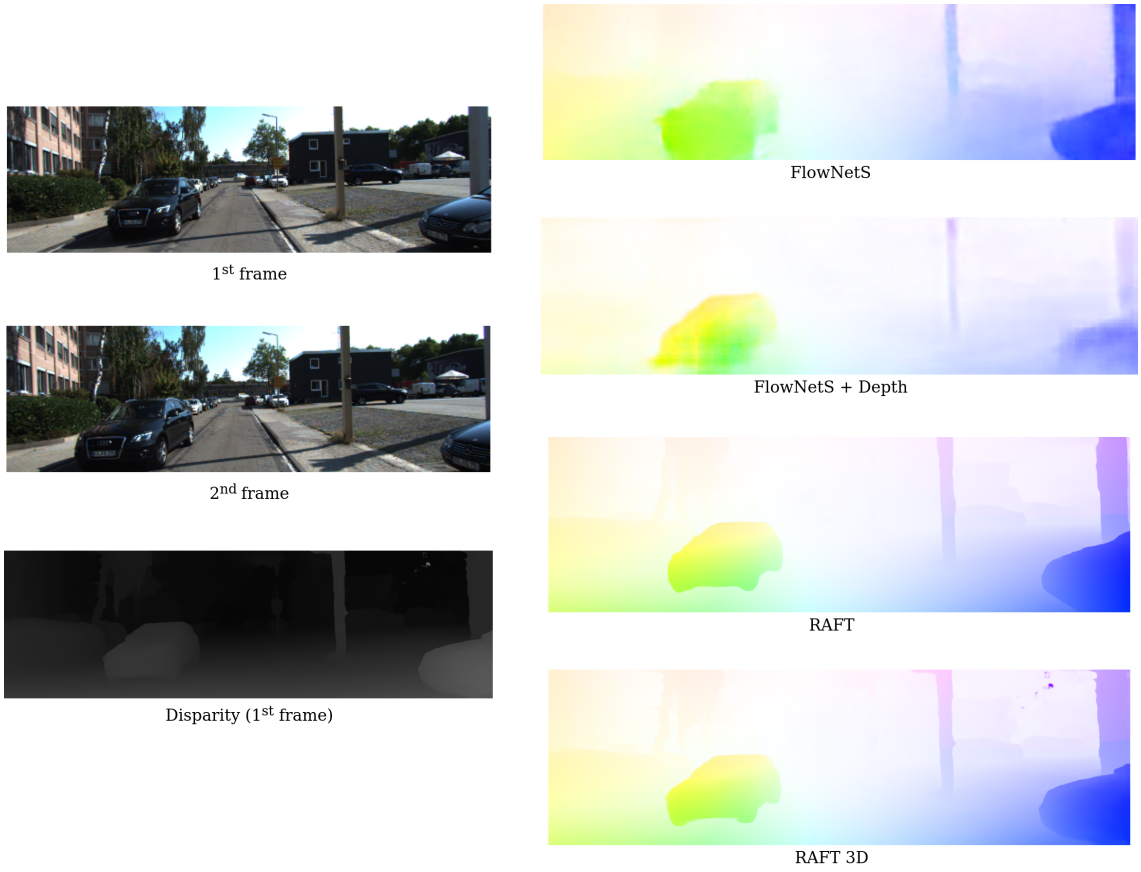


Figure 4.2: Optical flow estimates given by different architectures. These are computed between the first and second frames. In this case, the disparity image aids FlowNetS + Depth. This network is better, than FlowNetS, at obtaining the shape of the upcoming vehicle, in terms of optical flow vectors.

The inspection of Figure 4.2, where the upcoming car is moving, allows inferring each network's ability to estimate the motion of each moving obstacle. As expected, the original FlowNetS had the worst approximation of the the moving vehicle's boundaries presenting a boxy shape, also getting affected by its shadow. In contrast, the same architecture with added depth is much better at estimating the borders around the object, supporting the claim that the added depth indeed helps in optical flow estimation. RAFT seems quite accurate at inferring the object's shape by its movement. This happens because of the context encoder present in the architecture. The RAFT-3D model seems to work as well as RAFT for the purpose of detecting the optical flow of the moving object against the background. It is also interesting to note some noise in the top right corner of the optical flow estimate that seems to have an origin in the disparity image. The methods where the depth was added as input, might not have as good results as the others because the depth is also estimated and not 100% reliable, possibly introducing some unwanted errors. One way to

possibly make the depth estimation more reliable could pass by using a different sensor, a LiDAR. Using the accurate depth perception of a LiDAR sensor, fusing the depth measures with the images obtains accurate depth measures for a sparse set of pixels. Although more accurate LiDAR generated point clouds tend to be sparse. Because of this, introducing the LiDAR acquired depth in RAFT-3D or FlowNetS would imply substantial changes to the networks, possibly making the design of a new model a better idea. Furthermore, a dataset containing many images, LiDAR collected point clouds, and optical flow ground truths does not currently exist.

4.3 Moving object segmentation

In this section, the moving object segmentation network will be subjected to various experiments to validate its efficiency. The segmentation network is based on the U-Net architecture and was trained using different optical flow estimators as inputs. For this effect, the methods taken into account were FlowNet+Depth, and RAFT. The dataset used to train the segmentation network comprises a total of 12919 images, where 80% were used for training, 10% for validation, and the remaining 10% for testing. The training parameters used for both cases are presented in Table 4.3:

Table 4.3: U-Net training parameters

Batch size	5
Epoch #	100
Start Learning Rate (LR)	$1e^{-3}$
Optimizer	Adam
Loss	Cross entropy loss
LR scheduler	Reduce LR on plateau

As already mentioned, the network's inputs are 3-channel images, where two channels are composed of the u-flow and the v-flow, and a third channel is the disparity of the corresponding image.

The trained network was evaluated on three sequences from the raw KITTI odometry dataset that presented different challenges: sequences 11 and 57 present scenes where the cars have a perpendicular movement relative to the sensor. On the contrary, in sequence 14, most of the moving objects approach the ego-vehicle on the left lane in the opposing direction of movement. This test scenario presents a more challenging testing condition as the optical flow vectors will not be as pronounced when these vehicles move in the same direction as the background. Sequence 14 also presents illumination variations. In optical flow the intensity is considered constant between two sequential frames, if a scenario presents a lot of shadows or tunnels, this assumption is not true. This could impact the developed method, as optical flow is the principal input component. Representative frames from these scenarios are presented in Figure 4.3.

It is possible to use several metrics to evaluate the segmentation procedure. The dice-score was already discussed in Section 3.4. Other metrics, such as pixel accuracy and IoU, could be used. Pixel accuracy is the simplest one, and it is defined as the percentage of correctly estimated

(a) (b)

Figure 4.3: Representative frames of the moving object segmentation testing sequences. In (a) the moving vehicles movement is perpendicular to the camera. In (b) most vehicles move in the opposite direction, of the sensor.

pixels in an image; this metric is by no means perfect. In a large image where the ground truth class only covers a small percentage of the area, the difference in pixel accuracy between a perfect segmentation and an estimate where no classes are detected would be minimal, giving a false sensation of good results. The IoU is a more accurate measure used in other computer vision areas such as image classification. It is defined as the area of the intersection between the ground truth (GT) and the estimates (E), divided by their area of overlap. The results, in terms of IoU and dice-score, are presented in Table 4.4, considering the training with both optical flow estimations (FlownetS+Depth and RAFT).

Table 4.4: Moving object segmentation results.

Seq. #	Dice-score		IoU	
	FlownetS+Depth	RAFT	FlownetS+Depth	RAFT
11	0.43	0.59	0.39	0.53
14	0.38	0.48	0.31	0.40
57	0.44	0.78	0.35	0.68

The presented results show that the network obtains better IoU and dice-score when the RAFT optical flow is used. As previously shown, by analysis of Figure 4.2, RAFT is better than FlownetS+Depth at inferring different moving objects and their respective borders. It is also important to say that the ground truth for the moving obstacles is not perfect as some only have their corresponding bounding box, as in Figure 4.4.

As expected, the results were better in sequence 57, where most vehicles move perpendicular to the camera frame. The inferior results in sequence 11 are also explainable, as this sequence has a larger sub-set of frames with no moving objects. According to the dice-score formula, Equation 3.2, the score is undetermined if there are no detections in the ground truth or estimation. The result is correct and should be attributed to a dice-score of 1. The problem in sequence 11 relies on the fact that there are a few FP detections that significantly lower the scene score, giving a score of zero to frames with some false detections when there are no moving objects at all. This discussion could also be extended to the IoU metric, even a small area of a FP detection will consider the image IoU coefficient as zero. It is also important to point out that in each frame, the percentage of the image covered by moving objects is low. If the evaluation scenario is inverted,

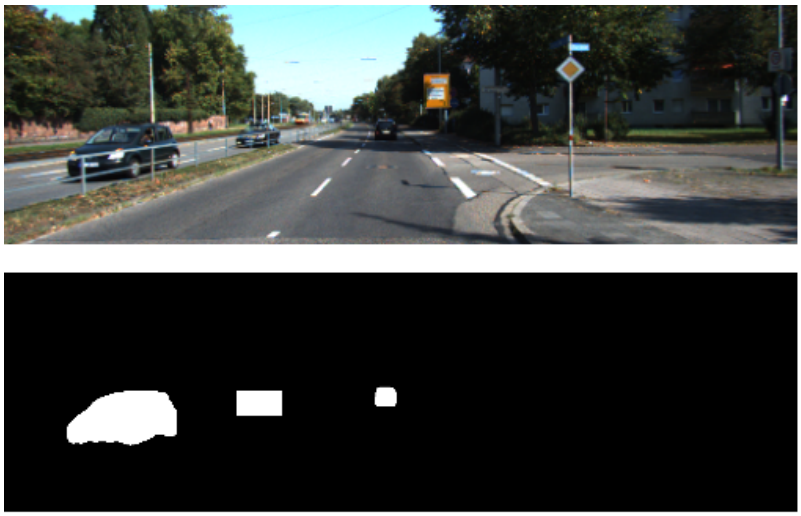


Figure 4.4: Example of a non-perfect ground truth mask, as the moving car only has the corresponding bounding box, and not the actual shape of the vehicle.

using the same metrics, measuring for the static background instead of the moving foreground, the results would be skewed and far better. For example, the presence of FP detections would be smaller as the area of pixels considered static, even though they are moving, would be minimal in each frame. In this way, all metrics would give results close to 100% accuracy, and it would not be as interesting to compare the results. This also notes the importance of understanding how each metric works to assess the results fairly. In the future application scenario proposed in this document, a few false-positive detections are not problematic. As the method relies on a dense optical flow estimation with almost half a million vectors, excluding a small percentage will not impact the future accuracy of the odometry system. The trained network successfully detects moving objects, struggling with false positives and far away cars where the optical flow is just not precise enough.

Figure 4.5 shows some examples of the moving object segmentation in various scenarios presenting its strengths and weaknesses, using the RAFT optical flow as input. This figure allows analyzing the IoU visually. The ground truth for moving objects is presented in green, and the estimates are colored in red. The intersections of both (TP) are highlighted in white. In image (a), the cars move perpendicular to the sensor. In this case, the network is able to detect all moving vehicles, even though the detections present a small "overflow" of false positives. In (b) the network can also predict every moving vehicle in the same and opposite direction of the sensor movement. Some false positive pixels are noticeable between the three cars in the distance. This happens because, with distance, both optical flow and disparity estimates become less accurate. As all three cars have similar velocity and directions of movement, U-Net has a hard time individually inferring each car's movement. In images (c) and (d), the scenario is similar to (b), as the network has difficulties at inferring the movement of objects far away while accurately segmenting the ones that stand close to the sensor. In (e), the case of a false positive pixel "overflow"

happens again. One moving object is partially occluded, and the network segments the entirety of the vehicle even though it is occluded, which in the future application scenario is not a problem. Lastly, in the image (f), U-Net can identify a far-away moving object. It happens because the rest of the frame is only composed of a static background. In this case, the method can easily detect the different flow vectors caused by the vehicle's motion.

Figure 4.5: Example of moving object segmentation in several scenarios. Green is the ground truth, red the estimates and the true positives are marked in white.

In conclusion, the trained U-Net architecture successfully identifies and segments moving objects. The primary limitations seem to rely on the accuracy of the inputs: optical flow, and disparity rather than on the well-known and tested U-Net architecture. This evaluation showed a successful proof of concept. With some manipulation regarding the underlying network architecture, added with the additional time to further explore training regiments, this architecture could be fine tuned to become even more precise. Noting the visible limitations of the current state of this design, struggling with false positives and detecting far away moving obstacles. These are not unsurpassable limitations, false positives only remove a small percentage of many optical flow vectors that can be used to estimate the vehicle's ego-motion. Given the increased distance, their disparity is less accurate, so the outlier rejection mechanism will not consider most of these points. Therefore the performance of this moving object segmentation tool is satisfactory to be used in further parts of the architecture.

4.4 Odometry evaluation

All of the necessary components for odometry estimation have been obtained and evaluated at this stage. It is necessary to combine the multiple modules and evaluate the odometry algorithm

as a whole. It is essential to consider that the errors from previous modules, such as the optical flow error, will significantly impact the performance of the algorithm at this stage. The evaluation procedure will be composed of two main parts. The system will be tested in scenarios with few dynamic objects to evaluate the general performance of the odometry estimation. Several optical flow networks will be used as inputs to infer which is the best option for this step. The algorithm will be evaluated in scenarios containing many dynamic objects in a second step. To evaluate the effectiveness and how the dynamic masking procedure affects the odometry estimation, the algorithm will be tested with and without this feature, using the same optical flow estimator, for a fair comparison. Two datasets were used for evaluation. Some sequences of the original KITTI dataset will be used to evaluate the general performance in static environments. Almost all algorithms have their performances listed on the KITTI odometry benchmark; this will be useful to compare the results of the developed system against the state-of-the-art. Because KITTI only has a few sequences where the presence of moving objects can be considered significant, to evaluate the performance in highly dynamic environments, the evaluation will be extended to KITTI-360 (identified with "_360"). This dataset is the second iteration of KITTI but has longer sequences in a wider variety of scenarios.

It is necessary to evaluate the execution time. As the entire method is composed of multiple parts, each individual module should be taken into account. In the evaluated scenarios, the disparity network, PSMNet, is able make an estimation every 100 ms. In terms of optical flow, three were tested, and have different run times: FlownetS (6,6 ms), RAFT (100 ms) and RAFT-3D (283 ms). The implemented dynamic segmentation network has an inference time of 43 ms per image. Finally, with the parameters used in the other tests (130 RANSAC iterations) has a run time of 150 ms. Considering only one GPU and that the calculations are done in a sequential approach, the total run time varies between 300 ms and 576 ms depending on the used optical flow method. It is important to take in mind that the used code was written in Python and could be further optimized, especially the ego-motion estimation algorithm. The disparity and optical flow estimators can also be substituted or redesigned to work more efficiently. For example FlownetS has suitable run times for real-time operation but the performance is significantly lower when compared to RAFT.

The metrics used to evaluate each sequence should also be defined and explained, as there are several ways to evaluate the odometry estimations. The KITTI odometry evaluation toolbox [85] was employed and extended to the KITTI-360 dataset. The estimates obtained by the odometry algorithm in the form of $[T_X, T_Y, T_Z, \Omega_X, \Omega_Y, \Omega_Z]^T$ should be transformed into a sequence of rigid body transformations, composed of a rotation and a translation, all relative to the first pose, $P_1, ..., P_n \in SE(3)$.

The simplest evaluation metric is the Absolute Trajectory Error (ATE). The difference from the prediction to the ground truth is calculated for all axis, over an entire sequence. The ATE is defined as the root-mean-square error of these differences, as in Equation 4.2, where n is the number of individual poses.

$$ATE = \sqrt{\frac{(x_{gt} - x_{pred})^2 + (y_{gt} - y_{pred})^2 + (z_{gt} - z_{pred})^2}{n}} \tag{4.2}$$

As this method only presents frame-to-frame ego-motion estimation and does not include any loop closure procedure, even a singular large enough error estimate could propagate throughout the entire procedure. ATE does not take this into account. A relative metric that only considers the transformation between consecutive poses should also be used. Taking the predictions from the algorithms, in the form of rigid body transformations relative to the first pose, $P_1, ..., P_n \in SE(3)$, and the correspondent set of ground truth poses, $Q_1, ..., Q_n \in SE(3)$, the Relative Pose Error (RPE) between two consecutive poses is defined as:

$$RPE_{i,i+1} = (Q_i^{-1} Q_{i+1})^{-1} (P_i^{-1} P_{i+1}) \tag{4.3}$$

To evaluate the RPE over an entire sequence in terms of meters, only the error translation components of the rigid transformation $(\delta x_{i,i+1}, \delta y_{i,i+1}, \delta z_{i,i+1})$ are taken into account and the RMSE is used, Equation 4.4:

$$RMSE_{RPE} = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n-1} (\delta x_{i,i+1}^2 + \delta y_{i,i+1}^2 + \delta z_{i,i+1}^2)} \tag{4.4}$$

A similar metric to $RMSE_{RPE}$ also used for evaluation consists on calculating the euclidean distance of relative translations, obtained in Equation 4.3, between the prediction and the ground truth. The mean euclidean distance of an entire sequence is obtained, as shown in Equation 4.5.

$$Mean_{RPE} = \frac{1}{n-1} \sum_{i=1}^{n-1} \sqrt{\delta x_{i,i+1}^2 + \delta y_{i,i+1}^2 + \delta z_{i,i+1}^2} \tag{4.5}$$

The last used metric, sub-sequence translation error (Sub_{err}), involves segmenting the sequence into equal sizes and then use equations 4.3 and 4.4 to calculate the translational error between the first and last frame of each sub-sequence. In this case the used sets were divided using lengths [100, 200, 300, 400, 500, 600, 700, 800] meters. Then the percentage of translational drift is computed for each of the cases, allowing several measures for the same sequence in terms of different distance sub-sequences.

The odometry algorithm was tested on a total of eight sequences to evaluate the results using different optical flow estimates, namely RAFT and RAFT-3D, and to evaluate the efficacy of the dynamic mask. Table 4.5 contains all gathered results from the tested scenarios. Some particular cases in both static and dynamic environments are shown in more depth in Figures 4.6-4.8.

Regarding a comparison between the algorithm's performance when using RAFT against RAFT-3D, both without dynamic masks and in static scenarios. By analyzing the first half of Table 4.5 it is not immediately clear which optical flow performs in the best. For example, in sequence 00, it is evident that RAFT w/o mask outperforms RAFT-3D. To understand the differences more deeply, Figures 4.6 and 4.7 have some graphs to help visualize what happens in sequences 00 and 07.

In Table 4.5, regarding sequences 00 and 07, RAFT consistently outperforms RAFT-3D. In fact, regarding the relative pose estimates of sub-figures 4.6a-4.7a, it is visible that, while not

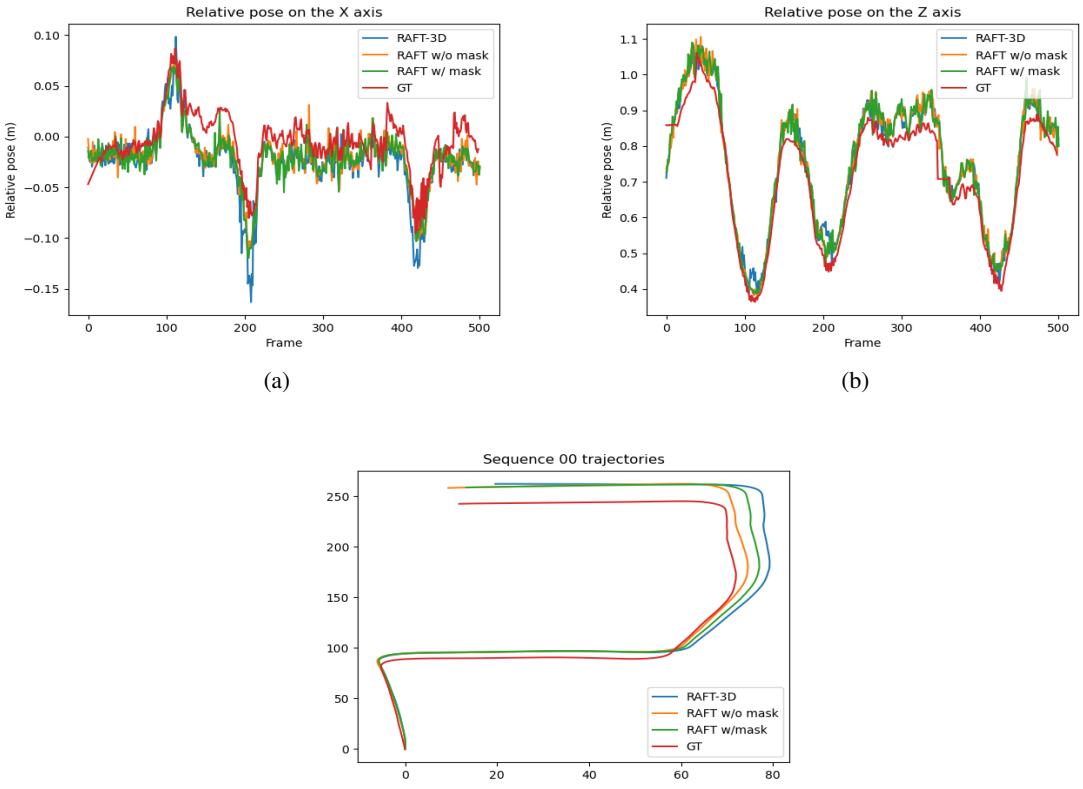
Table 4.5: Ego-motion estimation results.

Seq. (no. frames)	$Sub_{err}(\%)$	$ATE(m)$	$RMSE_{RPE}(m)$	$Mean_{RPE}(m)$
00 (500)	3.69	3.85	0.074	0.063
	4.26	3.27	0.074	0.063
	5.25	4.09	0.079	0.065
03 (800)	12.9	4.58	0.133	0.119
	13.47	5.75	0.135	0.124
	11.76	7.61	0.105	0.094
06 (1100)	5.84	23.25	0.139	0.127
	5.6	25.21	0.130	0.119
	7.42	20.49	0.166	0.147
07 (500)	3.41	6.44	0.077	0.068
	3.52	6.69	0.078	0.069
	5.27	13.39	0.093	0.085
04* (270)	13.81	20.86	0.293	0.260
	13.329	20.49	0.28	0.250
	11.71	20.12	0.149	0.130
07_360* (335)	12.94	18.37	0.256	0.146
	12.144	26.962	0.217	0.135
	7.03	8.34	0.202	0.145
10_360* (129)	8.278	5.703	0.405	0.323
	7.067	5.77	0.376	0.283
	3	2.5	0.145	0.124
11_360* (130)	5.78	3.38	0.150	0.133
	5.38	3.23	0.140	0.126
	9.29	4.62	0.213	0.196

Color code:

RAFT w/o mask	RAFT w/mask	RAFT-3D

*- scenarios with a substantial amount of dynamic objects



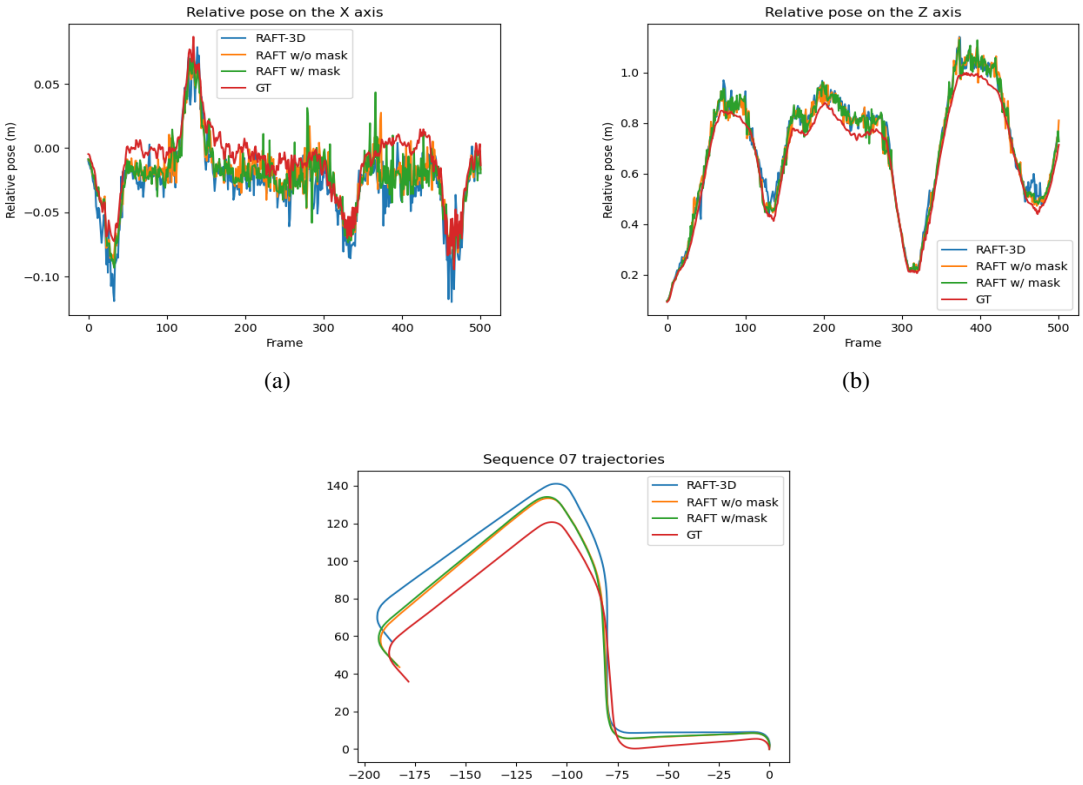
(a) (b)

(c)

Figure 4.6: Graphs relative to sequence 00 (static).

perfect, RAFT always tends to be closer to the ground truth than RAFT-3D and also proves to be significantly less noisy, this translates into a lower RPE. In these cases, the lower RPE from RAFT also translated into a lower ATE, which can be observed in sub-figures 4.6c and 4.7c. It is notorious that RAFT's trajectory stays closer to the ground truth than RAFT-3D. However, this is not always the case. In Table 4.5, sequence 06 shows that RAFT has a lower relative error while RAFT-3D has a lower absolute error and vice versa on sequence 03. Even if the mean RPE is low across an entire sequence, a single large error could completely ruin the ATE measure. For example, if the rotation varies excessively on a single estimate, the following trajectory will take a completely different path. Odometry is an integrative process accumulating all the previous errors into the new estimates, this is called drift. This problem is almost unsurpassable if one only considers every two sequential frames. Therefore it is not only good enough to have method which has a low mean relative pose error; it is also necessary to be robust enough, minimizing error spikes. Usually, loop closure procedures are employed, and a map of the traversed environment is maintained. Suppose the method detects that the place it is passing was already mapped; it can perform a loop closure procedure, declaring constraints on the estimate.

From this test, it was also possible to understand that, as expected, this method performs better in more urban environments with close quarters, where the disparity and optical flow estimates are



(a) (b)

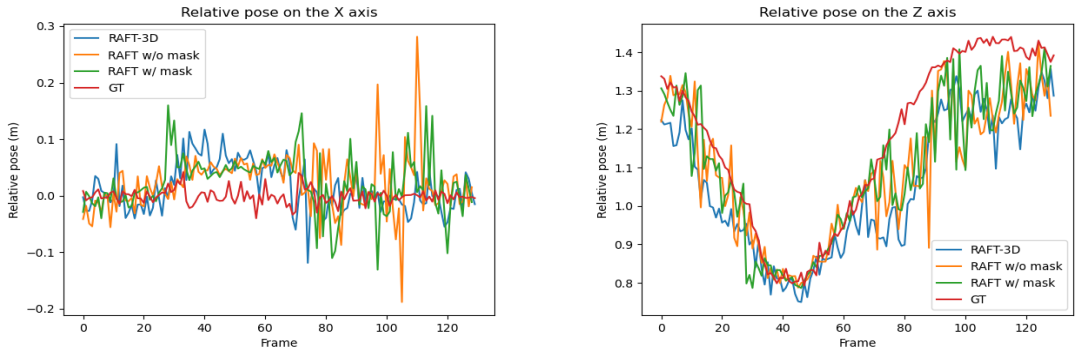
(c)

Figure 4.7: Graphs relative to sequence 07 (static).

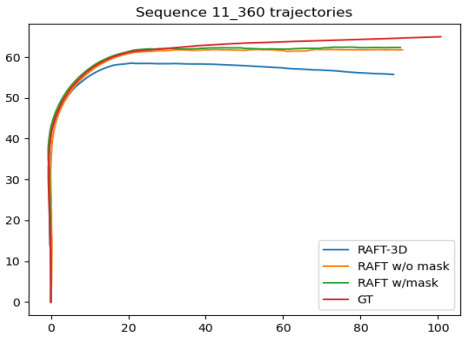
more accurate. For example, comparing the better scoring sequence 00 against the lower scoring sequence 06, Figure 4.9, it is evident that sequence 00 has narrower streets, and the disparity conveys more information. Sequence 00 also has more features that are useful for the optical flow estimators, while sequence 06 is composed of a larger open space with fewer good features.

Now that a general evaluation of the ego-motion method in static scenarios has been established, it is time to evaluate the dynamic object mask's impact. As previously explained, the objective is to detect and segment the dynamic objects in the scene to remove their influence from the odometry estimation procedure. The results using the optical flow estimator RAFT with and without mask will be analyzed, taking a closer look on the dynamic environments scored in the bottom half of Table 4.5.

When taking the dynamic objects into account, the algorithm's performance increases across all dynamic scenarios. The case that does not seem to have such an obvious improvement is sequence 07_360. Nonetheless, one can check that the mean and RMSE relative errors decreased, so this case boils down to the already explained drift phenomenon and the effect of a singular bad estimate. It is also important to note that the method without the dynamic masking operation still performs outlier removal via RANSAC. While one cannot be sure that RANSAC will eliminate every dynamic object, it still acts as a 'soft' dynamic object remover. If this procedure was



(a) (b)



(c)

Figure 4.8: Graphs relative to sequence 11_360 (dynamic).

(a) (b)

(c) (d)

Figure 4.9: Comparison between sequence 00 ((b) and (d)) against sequence 06 ((a) and (c)). The disparity of sequence 00 conveys more precise and relevant information.

excluded, the evaluation would not be a fair one-to-one comparison, as the results without any outlier removal tend to be highly erroneous. To remember how the masking operation is done, first, U-Net computes the masks of all dynamic objects in each frame. The correspondent flow

vectors and disparity values of the pixels labeled as dynamic will not take part in the least-square optimization procedure.

To further understand if this procedure works, one example, sequence 11_360, will be analyzed in more detail. In a few words, in sequence 11_360, the ego-vehicle traverses a calm road with no other cars, and around frame number 65, a convoy of four vehicle cross with the ego-vehicle in the opposite direction. As the road is narrow, these are expected to affect the performance of the ego-motion algorithm. The scenario is exposed in Figure 4.10

Figure 4.10: Segmentation of moving vehicles, in blue, in sequence 11_360.

The overlaid dynamic object mask indicates the successful detection of the two closest cars, which could affect the odometry estimation. By analyzing the graphs in Figures 4.8a and 4.8b, from frame number 65 to frame 120, which correspond to the frames where the objects were visible in the images. It is simple to check that the error of the relative poses, either on the X or Z axis, is considerably lower with the mask. This sustains the point that the dynamic objects indeed affect the ego-motion estimation and that the designed odometry system considers them, to reduce the ego-motion estimation error.

It was proven that the dynamic masking procedure struggles with FP detections, especially in moments were no moving objects are in frame. It is necessary to prove that these FP do not hinder the performance of the odometry algorithm in static scenarios, by removing important points used for calculations. Inspecting the static portion of Table 4.5, it is clear that the results between RAFT w/mask ans w/o mask are very similar in terms of RPE and ATE. So it is fair to conclude that the dynamic masking procedure contributes to better performance in dynamic scenarios while not compromising in static ones.

To compare the performance of the developed algorithm against the state of the art some tests were done on the KITTI dataset. The selected method is an open-source implementation of SOFT-SLAM [25], which occupies a top 20 position on the KITTI odometry benchmark. For a fair comparison, like the developed work, SOFT-SLAM is a stereo visual odometry algorithm, and the gathered results do not have any loop closure or mapping methods enabled. Table 4.6 gathers the results against RAFT w/mask on a few sequences.

It is obvious that SOFT-SLAM outperforms the developed odometry system, consistently across most sequences, with the exception of sequence 07 where the developed method is better. To better compare both methods, Figure 4.11 depicts relative and absolute trajectories across

Table 4.6: Comparison between developed method and SOFT-SLAM.

Seq. (no. frames)	$Sub_{err}(\%)$	$ATE(m)$	$RMSE_{RPE}(m)$	$Mean_{RPE}(m)$
00	4.26	3.27	0.074	0.063
(500)	2.95	5.37	0.032	0.028
03	13.47	5.75	0.135	0.124
(800)	4.23	3.45	0.039	0.032
06	5.6	25.21	0.130	0.119
(1100)	2.61	6.43	0.046	0.039
07	3.52	6.69	0.078	0.069
(500)	8.72	14.37	0.075	0.070
04*	13.329	20.49	0.28	0.250
(270)	2.70	5.57	0.048	0.043

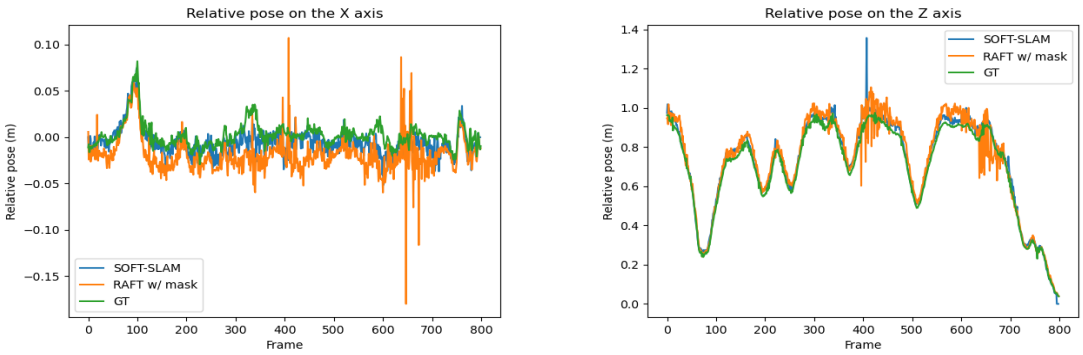
Color code:

RAFT w/ mask	SOFT-SLAM	

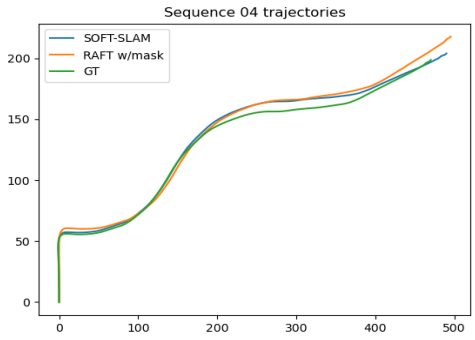
*- scenarios with a substantial amount of dynamic objects

sequence 04. It in Figures 4.11a and 4.11b is is noticeable that SOFT-SLAM has less noisy frame-to-frame estimates when compared to the developed method, still the better method also has some places where it struggles and shows some error spikes. Nonetheless SOFT-SLAM is consistently closer to the ground truth and therefore the trajectory in Figure 4.11c is also more accurate. It is also fair to say that despite being inferior results the proposed method achieves respectable trajectories.

The results are in accordance to the state-of-the-art literature review were the classical methods outperform the ones that make the use of deep learning methodologies. Nonetheless, the developed work shows promising results that with some improvements could reach the state-of-the-art performance. It is positive, that this innovative hybrid-based approach is able to achieve results close to other algorithms that are tried and tested. The used approach of leveraging the power of deep learning, on disparity and optical flow, with proven knowledge-based techniques to recover the estimation shows great promises and might be the next step towards improving deep learning works in the area of visual odometry.



(a) (b)



(c)

Figure 4.11: Comparison between SOFT-SLAM and the developed method on sequence 04.

Chapter 5

Conclusions and Future Work

This dissertation proposes a dynamics-aware visual odometry system explicitly designed for autonomous driving. Ego-motion estimation is a critical aspect of any autonomous moving system. The need to self-localize is crucial so that the robotic system can perform any other task, such as trajectory planning. The driving scenario is composed of particular limitations regarding the problem of visual odometry. The main focus of this dissertation involved removing the influence of other moving obstacles in the scenario. The proposed work employs a novel architecture in the area of VO, integrating deep-learning and knowledge-based algorithms into the framework. This work combines the strengths of both areas. In optical flow and depth estimation, deep-learning approaches have surpassed the classical ones, but in pose estimation, it has not been able to catch up. Therefore this dissertation contributes with a novel architecture, joining both approaches in a way not yet discussed in the state-of-the-art. Regarding VO, the base architecture shows good early results, as low as 3.6% in sub-sequence translation error with about 63 centimeters of relative pose error averaged across an entire sequence of 500 frames. Although not up to par with top approaches, struggling in specific conditions such as wide open spaces, it shows great promise for a standard frame-to-frame estimation method. This document also exposed a novel method for segmenting moving objects, quite different from the state-of-the-art approaches. While suffering some downfalls in specific scenarios, this method segments moving objects that could be a source of errors in VO algorithms. The segmentation method was designed with this odometry algorithm in mind, as it uses the optical flow and disparity estimations also needed in the ego-motion estimation step. Nonetheless, it could be easily implemented into other visual odometry works, specifically stereo ones, as it does not need more information than sequential frames from a stereo camera. Joining the moving object detection module with the standard VO algorithm did show a 14% improvement in specific scenarios. Of course, this depends on the specificity of each scenario, as both modules have diverse limitations. Despite every limitation, this algorithm shows a successful proof of concept for a novel hybrid dynamics-aware visual odometry system based on optical flow and disparity estimation.

There are several ways the developed work could be improved and extended. For example, using a LiDAR, and this sensor's accurate depth measuring capabilities instead of a stereo camera

would significantly improve the depth estimates and, therefore, the accuracy of the VO algorithm. Using these LiDAR measures could also help improve the current state of deep optical and scene flow estimation. However, this would probably require designing a new network structure, and there are no available real-world datasets currently available. An interesting way of further exploring the already developed work involves the motion estimation of each moving object. Each moving object is already segmented, having a set of associated disparities and flow vectors. Using tracking mechanisms with the aid of a Kalman filter, it would be possible to track each moving object and apply the same principles to estimate their motion and trajectory. This would add value because such information would be essential to other mechanisms in autonomous systems, such as obstacle avoidance.

The developed work could be improved in the direction of a SLAM system to resolve the drift issue, as sequential errors tend to add up. Maintaining a local map and implementing a loop closure procedure would be the classical way to resolve this issue.

References

[1] I. Panagiotopoulos and G. Dimitrakopoulos, "An empirical investigation on consumers' intentions towards autonomous driving," *Transportation research part C: emerging technologies*, vol. 95, pp. 773–784, 2018.

[2] I. Barabás, A. Todoruţ, N. Cordoş, and A. Molea, "Current challenges in autonomous driving," in *IOP conference series: materials science and engineering*, vol. 252, p. 012096, IOP Publishing, 2017.

[3] M. Maurer, J. C. Gerdes, B. Lenz, and H. Winner, *Autonomous driving: technical, legal and social aspects*. Springer Nature, 2016.

[4] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, Ieee, 2004.

[5] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4104–4113, 2016.

[6] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011.

[7] Y. Alkendi, L. Seneviratne, and Y. Zweiri, "State of the art in vision-based localization techniques for autonomous navigation systems," *IEEE Access*, vol. 9, pp. 76847–76874, 2021.

[8] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*, pp. 404–417, Springer, 2006.

[9] S. A. Mohamed, M.-H. Haghbayan, T. Westerlund, J. Heikkonen, H. Tenhunen, and J. Plosila, "A survey on odometry for autonomous navigation systems," *IEEE Access*, vol. 7, pp. 97466–97486, 2019.

[10] R. Roriz, J. Cabral, and T. Gomes, "Automotive lidar technology: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[11] N. Jonnavithula, Y. Lyu, and Z. Zhang, "Lidar odometry methodologies for autonomous driving: A survey," *arXiv preprint arXiv:2109.06120*, 2021.

[12] M. Elhousni and X. Huang, "A survey on 3d lidar localization for autonomous vehicles," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1879–1884, IEEE, 2020.

[13] Y. Liu, N. Pears, P. L. Rosin, and P. Huber, *3D Imaging, Analysis and Applications*. Springer, 2020.

[14] A. Carballo, J. Lambert, A. Monrroy, D. Wong, P. Narksri, Y. Kitsukawa, E. Takeuchi, S. Kato, and K. Takeda, "Libre: The multiple 3d lidar dataset," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1094–1101, IEEE, 2020.

[15] M. Zhai, X. Xiang, N. Lv, and X. Kong, "Optical flow and scene flow estimation: A survey," *Pattern Recognition*, vol. 114, p. 107861, 2021.

[16] D. Sun, X. Yang, M.-Y. Liu, and J. 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*, pp. 8934–8943, 2018.

[17] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[18] D.-H. Kim and J.-H. Kim, "Effective background model-based rgb-d dense visual odometry in a dynamic environment," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1565–1573, 2016.

[19] W. Wang, J. Liu, C. Wang, B. Luo, and C. Zhang, "Dv-loam: Direct visual lidar odometry and mapping," *Remote Sensing*, vol. 13, no. 16, p. 3340, 2021.

[20] Y.-S. Shin, Y. S. Park, and A. Kim, "Direct visual slam using sparse depth for camera-lidar system," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5144–5151, IEEE, 2018.

[21] Y.-S. Shin, Y. S. Park, and A. Kim, "Dvl-slam: sparse depth enhanced direct visual-lidar slam," *Autonomous Robots*, vol. 44, no. 2, pp. 115–130, 2020.

[22] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE transactions on robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[23] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[24] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011.

[25] I. Cvišić, J. Ćesić, I. Marković, and I. Petrović, "Soft-slam: Computationally efficient stereo visual simultaneous localization and mapping for autonomous unmanned aerial vehicles," *Journal of field robotics*, vol. 35, no. 4, pp. 578–595, 2018.

[26] S. Wang, R. Clark, H. Wen, and N. Trigoni, "Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2043–2050, IEEE, 2017.

[27] N. Yang, L. v. Stumberg, R. Wang, and D. Cremers, "D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1281–1292, 2020.

[28] A. CS Kumar, S. M. Bhandarkar, and M. Prasad, "Depthnet: A recurrent neural network architecture for monocular depth prediction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 283–291, 2018.

[29] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, pp. 2938–2946, 2015.

[30] R. Zhu, M. Yang, W. Liu, R. Song, B. Yan, and Z. Xiao, "Deepavo: Efficient pose refining with feature distilling for deep visual odometry," *Neurocomputing*, vol. 467, pp. 22–35, 2022.

[31] S. Woo, J. Park, J.-Y. Lee, and I. S. Kweon, "Cbam: Convolutional block attention module," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 3–19, 2018.

[32] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, pp. 586–606, International Society for Optics and Photonics, 1992.

[33] J.-E. Deschaud, P. Dellenbach, B. Jacquet, and F. Goulette, "Ct-icp: Real-time elastic lidar odometry with loop closure," 2021.

[34] J.-E. Deschaud, "Imls-slam: scan-to-model matching based on 3d data," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2480–2485, IEEE, 2018.

[35] Y. Pan, P. Xiao, Y. He, Z. Shao, and Z. Li, "Mulls: Versatile lidar slam via multi-metric linear least square," *arXiv preprint arXiv:2102.03771*, 2021.

[36] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time.," in *Robotics: Science and Systems*, vol. 2, 2014.

[37] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, IEEE, 2018.

[38] J. Graeter, A. Wilczynski, and M. Lauer, "Limo: Lidar-monocular visual odometry," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 7872–7879, IEEE, 2018.

[39] H. Wang, C. Wang, C.-L. Chen, and L. Xie, "F-loam: Fast lidar odometry and mapping," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4390–4396, IEEE, 2021.

[40] M. Oelsch, M. Karimi, and E. Steinbach, "R-loam: Improving lidar odometry and mapping with point-to-mesh features of a known 3d reference object," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2068–2075, 2021.

[41] G. Wang, X. Wu, Z. Liu, and H. Wang, "Pwclo-net: Deep lidar odometry in 3d point clouds using hierarchical embedding mask optimization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15910–15919, 2021.

[42] Q. Li, S. Chen, C. Wang, X. Li, C. Wen, M. Cheng, and J. Li, "Lo-net: Deep real-time lidar odometry," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8473–8482, 2019.

[43] C. Zheng, Y. Lyu, M. Li, and Z. Zhang, "Lodonet: A deep neural network with 2d keypoint matching for 3d lidar odometry estimation," in *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 2391–2399, 2020.

[44] G. Chen, B. Wang, X. Wang, H. Deng, B. Wang, and S. Zhang, "Psf-lo: Parameterized semantic features based lidar odometry," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5056–5062, IEEE, 2021.

[45] D. Yin, Q. Zhang, J. Liu, X. Liang, Y. Wang, J. Maanpää, H. Ma, J. Hyyppä, and R. Chen, "Cae-lo: Lidar odometry leveraging fully unsupervised convolutional auto-encoder for interest point detection and feature description," *arXiv preprint arXiv:2001.01354*, 2020.

[46] Z. Li and N. Wang, "Dmlo: Deep matching lidar odometry," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6010–6017, IEEE, 2020.

[47] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "Epicflow: Edge-preserving interpolation of correspondences for optical flow," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1164–1172, 2015.

[48] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, "Deepflow: Large displacement optical flow with deep matching," in *Proceedings of the IEEE international conference on computer vision*, pp. 1385–1392, 2013.

[49] P. Dollár and C. L. Zitnick, "Structured forests for fast edge detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 1841–1848, 2013.

[50] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2758–2766, 2015.

[51] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2462–2470, 2017.

[52] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European conference on computer vision*, pp. 402–419, Springer, 2020.

[53] Z. Teed and J. Deng, "Raft-3d: Scene flow using rigid-motion embeddings," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8375–8384, 2021.

[54] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[55] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3061–3070, 2015.

[56] X. Liu, C. R. Qi, and L. J. Guibas, "Flownet3d: Learning scene flow in 3d point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 529–537, 2019.

[57] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *arXiv preprint arXiv:1706.02413*, 2017.

[58] W. Wu, Z. Wang, Z. Li, W. Liu, and L. Fuxin, "Pointpwc-net: A coarse-to-fine network for supervised and self-supervised scene flow estimation on 3d point clouds," *arXiv preprint arXiv:1911.12408*, 2019.

[59] X. Chen, S. Li, B. Mersch, L. Wiesmann, J. Gall, J. Behley, and C. Stachniss, "Moving object segmentation in 3d lidar data: A learning-based approach exploiting sequential data," *arXiv preprint arXiv:2105.08971*, 2021.

[60] T. Cortinhal, G. Tzelepis, and E. E. Aksoy, "Salsanext: fast, uncertainty-aware semantic segmentation of lidar point clouds for autonomous driving," *arXiv preprint arXiv:2003.03653*, 2020.

[61] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss, "Rangenet++: Fast and accurate lidar semantic segmentation," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4213–4220, IEEE, 2019.

[62] S. Li, X. Chen, Y. Liu, D. Dai, C. Stachniss, and J. Gall, "Multi-scale interaction for real-time lidar data segmentation on an embedded platform," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 738–745, 2021.

[63] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments.," in *Robotics: Science and Systems*, vol. 2018, 2018.

[64] P. Pfreundschuh, H. F. C. Hendrikx, V. Reijgwart, R. Dubé, R. Siegwart, and A. Cramariuc, "Dynamic object aware lidar slam based on automatic generation of training data," *arXiv preprint arXiv:2104.03657*, 2021.

[65] I. Alonso, L. Riazuelo, L. Montesano, and A. C. Murillo, "3d-mininet: Learning a 2d representation from point clouds for fast and efficient 3d lidar semantic segmentation," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5432–5439, 2020.

[66] X. Chen, A. Milioto, E. Palazzolo, P. Giguere, J. Behley, and C. Stachniss, "Suma++: Efficient lidar-based semantic slam," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4530–4537, IEEE, 2019.

[67] S. Pagad, D. Agarwal, S. Narayanan, K. Rangan, H. Kim, and G. Yalla, "Robust method for removing dynamic objects from point clouds," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10765–10771, IEEE, 2020.

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

[69] D. Yoon, T. Tang, and T. Barfoot, "Mapless online detection of dynamic objects in 3d lidar," in *2019 16th Conference on Computer and Robot Vision (CRV)*, pp. 113–120, IEEE, 2019.

[70] T. Y. Tang, D. J. Yoon, F. Pomerleau, and T. D. Barfoot, "Learning a bias correction for lidar-only motion estimation," in *2018 15th Conference on Computer and Robot Vision (CRV)*, pp. 166–173, IEEE, 2018.

[71] A. M. Pinto, A. P. Moreira, M. V. Correia, and P. G. Costa, "A flow-based motion perception technique for an autonomous robot system," *Journal of Intelligent & Robotic Systems*, vol. 75, no. 3, pp. 475–492, 2014.

[72] A. M. Pinto, P. G. Costa, M. V. Correia, A. C. Matos, and A. P. Moreira, "Visual motion perception for mobile robots through dense optical flow fields," *Robotics and Autonomous Systems*, vol. 87, pp. 1–14, 2017.

[73] A. M. Pinto, M. V. Correia, A. P. Moreira, and P. G. Costa, "Unsupervised flow-based motion analysis for an autonomous moving system," *Image and Vision Computing*, vol. 32, no. 6-7, pp. 391–404, 2014.

[74] P. Kumar Rath, A. Ramirez-Serrano, and D. Kumar Pratihar, "Real-time moving object detection and removal from 3d pointcloud data for humanoid navigation in dense gps-denied environments," *Engineering Reports*, vol. 2, no. 12, p. e12275, 2020.

[75] J. Kammerl, N. Blodow, R. B. Rusu, S. Gedikli, M. Beetz, and E. Steinbach, "Real-time compression of point cloud streams," in *2012 IEEE International Conference on Robotics and Automation*, pp. 778–785, IEEE, 2012.

[76] J. Huang, S. Yang, T.-J. Mu, and S.-M. Hu, "Clustervo: Clustering moving instances and estimating visual odometry for self and surroundings," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2168–2177, 2020.

[77] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[78] H. Rashed, M. Ramzy, V. Vaquero, A. El Sallab, G. Sistu, and S. Yogamani, "Fusemodnet: Real-time camera and lidar based moving object detection for robust low-light autonomous driving," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pp. 0–0, 2019.

[79] V. Vaquero, A. Sanfeliu, and F. Moreno-Noguer, "Hallucinating dense optical flow from sparse lidar for autonomous vehicles," in *2018 24th International Conference on Pattern Recognition (ICPR)*, pp. 1959–1964, IEEE, 2018.

[80] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[81] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. arXiv:1512.02134.

[82] J.-R. Chang and Y.-S. Chen, "Pyramid stereo matching network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5410–5418, 2018.

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

[84] A. M. G. Pinto, *Visual motion analysis based on a robotic moving system*. PhD thesis, Universidade do Porto (Portugal), 2014.

[85] H. Zhan, C. S. Weerasekera, J. Bian, and I. Reid, "Visual odometry revisited: What should be learnt?," *arXiv preprint arXiv:1909.09803*, 2019.