Master Thesis

Simultaneous Localization and Mapping in Dynamic Scenes

Author(s):
Barsan, Ioan Andrei

Publication Date:
2017

Permanent Link:

https://doi.org/10.3929/ethz-b-000202829 →

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision
and Geometry Lab

Simultaneous Localization and Mapping in Dynamic Scenes

Master Thesis

Ioan Andrei Bârsan

August 27, 2017

Advisors: Prof. Dr. Andreas Geiger, Peidong Liu
Supervisor: Prof. Dr. Marc Pollefeys

Department of Computer Science, Computer Vision and Geometry Group, ETH Zürich

Abstract

The task of dynamic scene understanding is an open problem in computer vision with numerous applications in fields such as autonomous driving. Many modern three-dimensional reconstruction pipelines do not account for the presence of dynamic objects in the environment, or simply treat all measurement associated with such entities as noise. Moreover, many reconstruction pipelines which rely on stereo input are unable to effectively deal with the noise typically associated with estimating depth from stereo.

We present a dense mapping system based on stereo cameras which is capable of robust operation in complex urban scenes. In addition to building a static map of its environment, the system also detects, tracks, and reconstructs the vehicles encountered within, while remaining memory-efficient thanks to a simple yet effective map pruning technique.

By leveraging instance-aware semantic segmentation, our system is capable of detecting and reconstructing not just actively moving objects, but also stationary objects with the potential of becoming dynamic, such as parked cars, which is highly desirable in many challenging higher-level tasks, such as urban path planning for autonomous vehicles.

We use the well-established InfiniTAM framework for performing volumetric reconstructions, adapting it to allow effective outdoor operation, and improving its scalability by reducing its memory requirements through a map pruning technique based on voxel garbage collection. Performing volumetric fusion for static objects and the environment map is straightforward, and only requires knowledge of the camera's egomotion, which is computed using visual odometry. Measurement registration for independently moving objects is achieved by tracking their 3D pose over time using sparse feature correspondences.

We perform a rigorous evaluation of our system's dense reconstruction capabilities, 3D pose tracking accuracy, run time, and memory footprint using sequences from the well-established KITTI odometry and tracking benchmarks.

Our system is capable of near real-time operation at roughly 2.5Hz, with the primary bottleneck being the instance-aware semantic segmentation, which is a limitation we hope to address in future work.

Acknowledgements

I would like to express my gratitude to my advisors, Peidong Liu and Prof. Andreas Geiger for giving me the opportunity to work on a fascinating topic in what is currently one of the most exciting areas of research: autonomous driving.

It goes without saying that I am also extremely thankful for my parents' constant support. I would not be where I am today if it had not been for them consistently pushing me to stay on track and achieve my potential, going back all the way to primary and secondary school. Thank you!

I would also like to offer my warm thanks to Alina and Cătălin Tufănaru for their continued advice, support, and guidance throughout my academic career.

Finally, I would also like to thank Xinyuan Yu for his technical insights into SLAM and dense mapping during the early stages of my thesis.

Contents

Chapter 1

Introduction

Over the course of the past two decades, the field of autonomous driving has undergone a dramatic evolution, propelling itself from laboratories and experimental tracks to real-world cities and highways. Promises of lower traffic-related fatalities, reduced congestion, and improved fuel-efficiency have attracted significant amounts of attention to this field, from academia and the industry alike.

Nevertheless, despite the field's rapid evolution, numerous open problems still remain on the path towards full automation. Examples include robustness to adverse weather conditions, the ability to understand and respond to nonverbal cues from pedestrians and law enforcement officers, as well as the task of motion planning in busy urban areas with large volumes of traffic, poor visibility, and unpredictable drivers.

While systems capable of reliable operation on highways and country roads have been available for decades, starting with the ALVINN project from 1989 [60], fully autonomous driving in urban environments is a considerably more challenging task. Reasons for this include reduced GPS accuracy due to interference, increased occlusion from buildings, vegetation, and traffic, disruptions cause by, e.g., construction work, and busier streets with unclear dynamics.

In this work, we direct our attention to the challenges associated with urban autonomy, focusing on the task of building detailed and reliable maps of dynamic environments such as busy roads and intersections. To this end, we develop *DynSLAM*, a dense mapping system capable of separately reconstructing both the static environment, as well as the vehicles within. This allows our system to accurately compute the static map, without the risk of corruption caused by moving objects. Moreover, the dense 3D models of the objects in the environment can be leveraged to improve the accuracy of components such as object tracking and motion planning.

1.1 Motivation

Despite their significant potential for positive social impact, the development and adoption of self-driving technologies is often still limited by the asso-

ciated costs, given that most such systems rely on expensive sensors, such as RADAR or LIDAR. In contrast, high-resolution color cameras can be orders of magnitude cheaper, while also providing richer information about the environment. Moreover, cameras have higher vertical resolution than sensors such as LIDAR, and are more robust in detecting reflective and semi-transparent surfaces. The field of camera-based mapping is therefore of great interest due to its potential to reduce the costs of autonomous vehicles, making them accessible to a wider audience.

While numerous existing mapping[1] systems are capable of operating within dynamic environments, they typically achieve this robustness by treating sensory input associated with dynamic objects as noise, and ignoring it in order to preserve the consistency of the static map (e.g., [72]).

Nevertheless, this process discards valuable information about the dynamic objects in an environment. While considerably more challenging than simple 2D tracking alone, building internal representations of the observed dynamic objects can substantially improve the performance of various other components:

- Densely reconstructing dynamic objects can allow the system to more accurately track them, by enabling it to reason about their exact 3D shape and predict their pose in new frames accordingly.
- Knowledge about the shape of the tracked objects can also improve the accuracy of motion planning, by enabling the autonomous agent to more accurately reason about the future occupancy of its environment.
- The availability of object reconstructions can enable realistic visualizations for, e.g., evaluating an autonomous vehicle's path planning process in a busy intersection, by using 3D models from the actual scene to represent other cars, as opposed to placeholder boxes or pre-defined CAD models.

While some existing SLAM systems do attempt to tackle the problem of reconstructing both static and dynamic objects, they either do it for robots constructing 2D LIDAR or RADAR-based maps [72, 6], do not produce dense reconstructions [65], or are unable to operate in (even near-) real-time [33, 36].

Other systems [2, 16] are constrained to low-dynamic environments, that is, environments which are expected to slowly change over time, but in which no directly observable motion is present. While being efficient at object identification through change detection, these systems are nevertheless still not suitable for high-dynamic applications such as autonomous driving.

Thorough comparisons of our work to the aforementioned systems, as well as to many others, will be presented in Chapter 2.

1.2 Contributions

The main goal of this thesis is the design, implementation, and evaluation of an outdoor dense mapping system capable of recovering both a map of

[1] This also includes Simultaneous Localization and Mapping (SLAM) systems, but in this work we focus on the mapping component. See §3.3 for more information.

the static parts of the environment, as well as separate 3D models of the potentially dynamic objects present in it, while operating in near real-time using stereo input.

Our contributions are therefore threefold. First, we design and implement an outdoor dense mapping pipeline by extending the existing InfiniTAM volumetric reconstruction system [38], originally designed for indoor operation.

Second, we use instance-aware semantic segmentation to detect potentially dynamic object instances such as cars, leveraging the sparse scene flow to reason about their 3D motion, and reconstruct them separately from the static map. Unlike most other dense reconstruction systems designed for operation in dynamic environments [33, 82, 39], our semantics-based approach ensures that the system is also aware of objects which are static, but likely to start moving in the near future. This opens up the potential for advanced applications such as urban motion planning and predictive obstacle avoidance.

Finally, we show how the memory efficiency and fidelity of the reconstructions can be improved by extending an existing map pruning technique based on voxel garbage collection to tackle reconstruction artifacts resulting from imprecise estimation of depth from stereo. To the best of our knowledge, our work is the first to leverage an active map pruning technique in the context of outdoor stereo-based reconstruction.

1.3 Outline

The remainder of this work is structured as follows: First, Chapter 2 covers related work, showcasing similar dense mapping pipelines, and highlighting the advantages and disadvantages of our system in relation to them. We also use this chapter to motivate our choices for certain components of our pipeline, such as using Efficient Large-scale Stereo [20] for disparity map computation, or the Multi-task Network Cascade neural network architecture [13] for instance-aware semantic segmentation.

Afterwards, Chapter 3 provides an overview of the core concepts used in this thesis, such as visual odometry, disparity estimation, dense mapping, and semantic segmentation.

Chapter 4 focuses on the core contributions of this work, describing how dynamic objects are detected, classified, tracked, and reconstructed on the fly. The second part of this chapter covers the map regularization used in our system, which is an enhanced version of the *voxel garbage collection* presented in [55], adapted for finer-grained map cleanup and improved scalability.

Chapter 5 presents a series of experiments on the KITTI odometry and tracking benchmarks [19], comparing various configurations of our system. We analyze the accuracy and runtime of our system, as well as various trade-offs which can be made, such as lowering the resolution of the input or the frequency of the semantic segmentation in order to increase the speed of the pipeline, albeit at the cost of reduced reconstruction accuracy.

Finally, Chapter 6 concludes our work, summarizing its contributions and results, and presenting several possible avenues for improvement, such as

adding support for loop closure detection and global map optimization, using multi-frame feature tracking for better 3D object tracking, or detecting moving objects in a more generic manner, without relying entirely on single-frame semantic information.

Chapter 2

Related Work

In the recent years, there has been growing interest in the field of autonomous driving, both in academia, and in the industry. Consequently, there is a vast body of recent and ongoing research in related fields, such as stereo vision, dense mapping, visual odometry, and object detection.

In this chapter, we provide an overview of the parts of this research most relevant to our work. We start by covering related work in Visual Odometry (§2.1), Depth from Stereo (§2.2), Dense Mapping (§2.3), Tracking (§2.4), and Semantic Segmentation and Object Detection (§2.5), motivating our choices for the respective DynSLAM pipeline components where applicable. Following this, in §2.6, we present several state-of-the-art dense SLAM systems designed to operate in dynamic environments, contrasting them with our own work, and highlighting the strengths and weaknesses of every approach.

2.1 Visual Odometry

ORB-SLAM2 [51] is a stereo extension of the original ORB-SLAM system [50], which leverages ORB (Oriented FAST and Rotated BRIEF) features for robust visual odometry, as well as other components such as relocalization and loop closure, which are not strictly related to our work. Thanks to the fact that ORB features are fast to compute and compare, their system is capable of running in real time on a standard CPU.

Geiger et al. [21] present `libviso2`, an efficient stereo-based visual odometry library which uses RANSAC to achieve robustness to outliers. Their approach is based on frame-to-frame tracking of simple blob and corner features. The objective function used for estimating the relative camera pose between two frames is formulated based on the sum of squared reprojection errors of 3D points triangulated from the previous stereo frame into the current left and right images. The authors leverage their proposed visual odometry method for building dense 3D reconstructions of urban environments, but without accounting for the presence of dynamic objects.

FOVIS [31] is a visual odometry method for RGB-D sensors based on FAST feature correspondences. It leverages both color and depth map features to estimate the camera egomotion, and is capable of running in real time on

low-end hardware. The system operates in a hierarchical manner, first computing a coarse estimate of the rotation using a simple direct image alignment method, followed by the sparse feature matching and final motion estimation stages.

While important, the visual odometry component is not essential for the scope of this work, given that our focus is on the mapping and reconstruction aspects. During prototyping and early experiments, we noticed that while global map consistency is a major challenge which we hope to address in future work, the frame-to-frame accuracy of the visual odometry does not have a strong influence on the local fidelity of the reconstructions. Concretely, we compared reconstructions computed using Geiger et al.'s [21] method to ones computed using ground truth pose information and found essentially no difference in quality, apart from the impact of drift, which only affects the map at a very large scale, without leading to loss of detail or significant distortion. Therefore, we have chosen to use `libviso2` in our pipeline, since it is a simple, well-established, and efficient visual odometry library. Moreover, while estimating the visual odometry, the library also computes the sparse scene flow, which can be used later in the pipeline for tracking objects in 3D. Further details about this method and how it is integrated in our system will be presented in §3.1.

2.2 Depth from Stereo

Considered to be one of the "classic" approaches to disparity estimation, Semi-Global Block Matching [30] casts the stereo matching task as a two-dimensional global optimization problem which it decomposes into multiple efficiently solvable one-dimensional sub-problems. While GPU-based implementations of this approach are capable of real-time operation, its accuracy is far from state of the art, suffering from heavy streaking artifacts and reaching a error rate of 10.86%[1] on the 2015 KITTI Stereo Benchmark [49].

The Efficient Large-scale Stereo (ELAS) system from Geiger et al. [20] first computes a triangulation of a robust sparse disparity map which it then uses to guide a second, dense phase. It is able to compute disparity maps on KITTI frames at roughly 8Hz, without requiring GPU acceleration. While only achieving an error rate of 9.72% on the KITTI 2015 Stereo Benchmark, ELAS is capable of producing very sharp disparity maps, which can lead to high-quality 3D reconstructions in dense mapping applications.

Güney and Geiger [26] propose Displets, a powerful stereo matching technique which reduces disparity estimation errors typically associated with reflective or transparent surfaces by leveraging semantic information and prior knowledge about object (car) shapes. While being very robust to challenges such as reflective and transparent surfaces, achieving a score of 3.43% on the KITTI 2015 Stereo Benchmark, this method is too slow for real-time operation, requiring 265 seconds to process one stereo pair.

[1] All reported errors use the D1-all metric, as described in the referenced paper and on the benchmark webpage, http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo.

A substantial amount of recent work in disparity estimation is based on neural networks. Žbontar and LeCun [79] train a neural network to compare patches for stereo matching. While achieving a high score on the 2015 KITTI Stereo Benchmark, with an error rate of just 3.89%, the method still produces artifacts around reflective and transparent areas, which are very common in autonomous driving applications. Moreover, inference time is still 0.8 seconds even when using the 'fast' parameter configuration, which is problematic for our near real-time operation goals. The SGM-Nets approach of Seki and Pollefeys [64] is able to outperform Žbontar and LeCun's method on the KITTI 2015 Stereo Benchmark, achieving 3.09% error, but requires 67 seconds to process one frame on the GPU.

The DispNet architecture presented by Mayer et al. [47] simply formulates disparity estimation as an end-to-end learning task, using an encoder-decoder architecture to directly compute dense depth maps from stereo pairs. They bootstrap their learning procedure using a novel synthetic dataset and finetune on KITTI training data, achieving a D1-all score of 4.34% on the 2015 Stereo Benchmark, but with a much shorter inference time than the other methods: just 0.06 seconds using an nVIDIA Titan X. Moreover, this method produces far fewer unwanted disparity artifacts near reflective or transparent surfaces than either MC-CNN or SGM-Nets.

Given our emphasis on real-time dense vehicle reconstructions, we prefer methods which are computationally fast and robust to reflective and transparent surfaces. In our work, we experiment with both Efficient Large-scale Stereo and DispNet. We choose these methods because of their fast run time, being able to run at 8Hz and 25Hz, respectively, their high performance on the 2015 KITTI Stereo Benchmark, and their open-source availability.

2.3 Dense Mapping

In this section, we present the state of the art in real-time dense scene reconstruction, focusing on methods operating in static environments. In §2.6, we will review systems which also take the dynamics of their environment into account.

Since the focus of this work is not on loop closure detection and global map consistency, we direct most of our attention to mapping systems relying exclusively on visual odometry or frame-to-model tracking for camera pose estimation.

2.3.1 Surfel-based Methods

First introduced by Pfister et al. [59] in the context of computer graphics, *surface elements* or *surfels* are a simple extension of a 3D point, augmented to also include a radius and a normal, akin to a small disc in space.

Due to their flexibility and potential for representing scenes at varying resolutions thanks to their radius parameter, surfels have been the subject of a substantial body of research in the field of 3D reconstruction [73, 29, 68, 35, 77, 48].

ElasticFusion [77] is a real-time surfel-based dense SLAM system capable of detecting both local and global loop closures on the fly, enforcing map consistency by means of a deformation graph. McCormac et al. [48] extend ElasticFusion with semantic mapping capabilities by also fusing semantic labelings of the 2D input frames into the 3D surfel-based representation in a Bayesian manner.

Wilkowski et al. [78] present an efficient method for generating, storing, and updating surfel maps by leveraging octree-accelerated frustum culling and replacing the traditional dense ICP-based tracking with a sparse variant.

2.3.2 Volumetric Methods

As opposed to the unstructured representations produced by surfel-based methods, volumetric ones implicitly encode a regular structure in the representation through their grid-based discretization.

One of the first methods using volumetric fusion for real-time scene reconstruction using a depth sensor was KinectFusion, the seminal work of Newcombe et al. [53], which sparked a large number of follow-up research improving its robustness [74] and scalability [75, 55, 80, 76, 38], or adding support for reconstructing non-rigid objects [52].

The DynamicFusion system from Newcombe et al. [52] also builds on KinectFusion, adding support for reconstructing deformable objects, such as human faces, by jointly estimating a canonical rigid model and a warp field in real time. MobileFusion [57] is a pipeline similar to KinectFusion that is capable of running in real time on a mobile phone, performing the reconstructions from purely monocular input.

2.3.3 Improving the Scalability of Volumetric Fusion

The original KinectFusion system, while being able to produce accurate, robust 3D reconstructions, was relatively limited in its scale, being constrained to desktop-sized environments. In the recent years, numerous methods have been proposed to increase the scale of the supported reconstructions to room-sized, or even street-sized scenes. Given the importance of supporting large-scale reconstructions in DynSLAM, we will now provide an in-depth overview of the possible approaches to achieving this.

Moving-volume representations One of the first works aiming to improve the scalability of KinectFusion was the Kintinuous system from [75]. While still relying on a fixed-size grid for performing the depth fusion, their system is capable of relocating the volume in physical space as the camera moves through a scene. Voxels which are left behind as the volume moves are converted to a mesh on-the-fly and streamed out to a global polygonal map. Follow-up work, such as [76], extends the system by adding loop closure detection and the ability to optimize the polygonal map for global consistency.

Octrees Some of the limitations of regular TSDF grids can be mitigated using an octree representation which allows the resolution of the final recon-

struction to adapt based on the quality of the input [80], and to avoid storing voxels corresponding to empty regions of space.

However, as pointed out by Nießner et al. [55], such approaches require deep tree structures for representing large scenes, which can negatively impact the performance of the GPU-accelerated components due to heavily branching code.

Voxel Hashing A different approach for efficiently representing large-scale maps is presented by Nießner et al. [55], and relies on a technique called *voxel block hashing*. Their method divides the scene into blocks consisting of $8 \times 8 \times 8$ voxels, which are stored in a compact manner in a hash table. By being able to avoid storing data for voxels belonging to empty areas of a scene, the voxel block hashing technique is capable of scaling to large environments, beyond what is possible with traditional dense grids.

Kähler et al. [38] present the InfiniTAM system, which builds on Nießner et al.'s work, adding support for visual-inertial tracking, while significantly improving the system's performance and portability. InfiniTAM is capable of running in real time (20Hz) on low-end devices such as an iPad Air 2, while still being able to take full advantage of dedicated GPUs, and run at over 1.1kHz on an nVIDIA Titan X. Follow-up work from the same authors [34] extends InfiniTAM with support for submaps and loop closure detection.

BundleFusion [12] also builds on the voxel block hashing framework developed by Nießner et al., adding online global camera trajectory optimization performed in a sparse-then-dense manner, with a first pass relying on sparse SIFT feature correspondences, followed by a second dense pass. The original framework is also extended to allow RGB-D frame de-registration, in addition to registration, effectively allowing entire frames to be deleted and re-integrated into the map in order to account for camera track adjustments, e.g., due to loop closure detection. Nevertheless, despite being able to produce state-of-the-art reconstructions, the system has very high computational costs, requiring two GPUs for real-time operation.

Due to its high performance and flexibility, as well as its open source availability and portable code, we have chosen to use InfiniTAM to perform the volumetric reconstructions for both objects and the static map in DynSLAM. In practice, we have observed it to perform much better than either Kintinuous [76] or ElasticFusion [77] on outdoor scenes with depth maps computed from stereo.

Sadly, the implementation of [34] had not yet been included in the public version of InfiniTAM used during the development of DynSLAM, so the additional features present in their work are not present in our final pipeline. Nevertheless, we are planning to integrate them into our system as part of future work. The InfiniTAM v3 technical report [61] published in July 2017 presents additional details on the implementation of these extensions.

(a) 3D object models produced from LIDAR and color data by [28].

(b) 3D object model produced from monocular video by [40].

Figure 2.1: Examples of 3D reconstructions produced by different tracking systems. Note that the former approach requires LIDAR, while the latter does not run online.

2.4 Tracking

With the increasing popularity of autonomous vehicle research, much work has been devoted to object tracking, due to its importance in obstacle detection and traffic understanding. Despite not being the primary focus of this thesis, tracking is still vital for the correct registration of color and depth measurements across multiple frames, a prerequisite for the volumetric reconstruction of dynamic objects presented in Chapter 4.

Held et al. [28] present an object tracking system leveraging both LIDAR and color information to track objects at very high frame rates. Unlike most classic approaches to tracking, the method also builds 3D models of the tracked objects, in order to increase robustness to occlusions and viewpoint changes. The tracking performance is ensured by the use of a novel data structure called *annealed dynamic histograms*, which allow the system to perform coarse-to-fine tracking by adjusting the sampling rate of the velocities from the state space in an efficient manner.

Lebeda et al. [39, 40] present a novel probabilistic approach for tracking and reconstructing dense 3D models of dynamic rigid objects from unstructured monocular RGB input, such as motor sports footage. Their key insight consists in casting the probabilistic reconstruction problem as a Gaussian Process inference task. The domain is defined as the angle of a particular point to the center of the object, with the radius being the dependent variable. From this formulation, a mesh can be extracted using MAP. Moreover, the variance at every point can serve as an indicator of the confidence in the accuracy of the reconstruction. This leads to a highly robust system capable of tracking and reconstructing arbitrary rigid objects in unstructured, cluttered, low-quality RGB video, even across shot cuts. This approach is, however, not without its limitations. First, performance-wise, the system is not real-time, requiring several seconds per frame. Moreover, in order to add color information to the reconstruction, the system needs to perform a second pass through the input sequence, which would be infeasible in a real-time application. Second, the framework is limited to reconstructing objects belonging to a star domain[2]. While this is a reasonable assumption for small automobiles, it will mean that

[2] Also known as a star-convex set, a star domain is a set in Euclidean space, centered around a point x_0 such that for every point x in the set, the line segment from x_0 to x is also in the set.

the system will be unable to correctly model more complex moving objects, such as pick-up trucks.

Sample reconstructions produced by the systems from Held et al. and Lebeda et al. can be seen in Figure 2.1.

Lenz et al. [41] present a stereo-based system which uses sparse scene flow to segment an environment into static and dynamic components. They compute a triangulation of scene flow points tracked over five frames, and discard edges between points with significantly different flow vectors, using Mahalanobis distances to account for the depth estimation error model. Moving objects are detected by performing a connected component analysis on the resulting scene flow graph. Following this, 3D bounding boxes are fit to the clusters of scene flow vectors, completing the tracking process.

The work of Choi [10] presents a multi-object tracking framework which leverages long term interest point trajectories for robustness to target drift, casting the frame association problem as a conditional random field.

Recently, there has been growing interest in applying neural networks to the field of object tracking. The Deep MANTA system presented by Chabot et al. [9] leverages deep neural networks to achieve a step up in terms of information richness, estimating not just the 3D bounding boxes of the tracked cars, but also their part structure, such as the size and location of their wheels, hood, top, etc., from monocular video. Byravan et al. [7] also leverage neural networks for directly estimating **SE3** transformations between point clouds.

2.5 Semantic Segmentation and Object Detection

Semantic segmentation and object detection are both well-studied fields of computer vision with numerous applications in autonomous robotics and scene understanding.

We now present a brief overview of recent work in both of these areas, motivating our choice to use the Multi-task Network Cascades (MNC) proposed by Dai et al. [13] in DynSLAM.

Semantic Segmentation The recent years have seen a significant increase in the accuracy of semantic segmentation methods, a success due in most part to the resurgence of neural networks. Examples include the works of Long et al. [43] and Badrinarayanan et al. [3], who augment traditional convolutional neural networks designed for single-output classification to perform per-pixel classifications by placing them in an encoder-decoder configuration. While the former focuses on adapting multiple classification architectures, such as AlexNet [37] and VGG16 [66], to the fully convolutional framework, the latter focuses on improving the smoothness of the outputs and the memory efficiency of both training and inference by reusing the max-pooling elements of the encoder in the decoder, thereby removing the need to learn dedicated upsampling layers.

Nevertheless, these methods are still computationally expensive, being unable to segment video streams in real time without significantly reducing

the resolution of the input. Paszke et al. [58] alleviate this issue with their ENet architecture, which is capable of processing 1280×720 input at 46.8 FPS, compared to SegNet, which only achieves 3.5 FPS on the same resolution and hardware. They achieve this significant performance boost by applying techniques such as performing feature map downsampling much earlier in their pipeline compared to other methods, using dilated convolutions to further increase receptive field sizes, and factorizing convolutional filters.

Object Detection The work of Uijlings et al. [69] is an object proposal generation method which starts with a superpixel segmentation of an image, merging the superpixels based on similarity cues until a particular threshold is reached. The bounding boxes of the resulting segments are computed and returned as generic object proposals.

The R-CNN proposed by Gurshick et al. [24] represents an extension of typical object proposal pipelines which also assigns a semantic class label to each proposal using a convolutional neural network (CNN). The proposals are generated by an external object proposal algorithm, rescaled to a standard size, and fed individually to the CNN for classification. The Fast and Faster R-CNN extensions [23, 63] improve the performance and runtime of the original method, with the former reducing training overhead by pre-computing image feature maps, and the latter further improving performance by also integrating the object proposal generation into the network architecture.

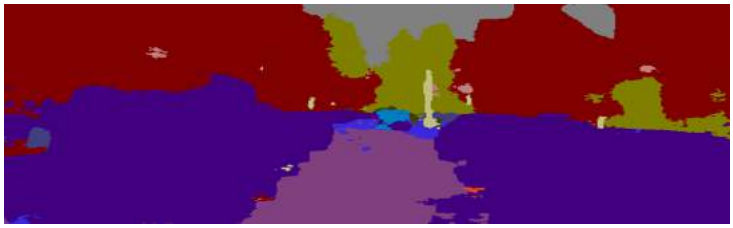
Instance-aware Semantic Segmentation Originally defined by Hariharan et al. [27] as *simultaneous detection and segmentation*, the problem of instance-aware semantic segmentation combines the semantic aspects of classic image segmentation methods with object detection, extending the notion of semantic segmentation to also include object instance information. In their work, Hariharan et al. present an extension of the R-CNN framework which operates on segmented object proposals, instead of box proposals. However, much like R-CNN, their work also requires numerous separate CNN passes at inference time, which leads to large computational costs, making the method unsuitable for real time operation (for instance, the object segment proposal component alone takes 30 seconds for a single image).

Dai et al. [13] formulate the semantic object detection problem as an end-to-end learning task, using a three-level cascade architecture to generate box proposals, to refine the proposals into pixel-wise instance masks, and, finally, to assign them a semantic label. Thanks to this unified framework, their method is able to perform the instance-aware segmentation task roughly two orders of magnitude faster than previous approaches such as [27], being able to segment and label object instances in less than 300ms on modern hardware[3].

In DynSLAM, it is necessary to both detect and classify objects, in order to track just those which have the potential to be dynamic (e.g., cars and pedestrians, but not tables), and to only attempt to reconstruct objects which are (mostly) rigid, such as cars, but not pedestrians.

[3]While their paper mentions inference times of 360ms, advances in GPU processing power since its publication have reduced it to less than 300ms at the time of writing.

(a) The original RGB frame from the left camera.



(b) Standard semantic segmentation produced using the classic SegNet [3] architecture. Note how there is no distinction between different objects of the same class (cars are dark blue), with the multiple cars being identified as a single blob of "car" pixels in the resulting segmentation.

(c) Instance-aware semantic segmentation computed by the MNC [13] architecture. Note that while the segmentation is not dense, and e.g., background pixels are not labeled, the different object instances are well-separated and labeled semantically.

Figure 2.2: A comparison between the output of a semantic segmentation pipeline and a semantic object detection one.

When performing 3D object reconstruction, accurate silhouettes of the target objects are necessary. Relying simply on bounding boxes for segmenting reconstructable objects would introduce too many artifacts in the reconstructed models, while also removing more information than necessary from the static map. Moreover, simple semantic segmentations of the input are not sufficient, since they do not carry with them information on specific object instances, causing separate objects belonging to the same class to be identified as a single blob. This limitation is highlighted in Figure 2.2.

We have therefore chosen to use the Multi-task Network Cascades (MNC) from Dai et al. [13] as the semantic object detection component of DynSLAM, due to the method's ability to segment and classify object instances, while also being near real-time in terms of computational efficiency.

Table 2.1: A brief overview of the features present in the related dynamic SLAM systems described in this section. A tick mark represents full support for a particular feature, while a tilde represents limited support. A system counts as building a dense map if it produces it on the fly, without requiring any post processing. We consider a system to be near real-time if it can process input in an online manner, and at a frequency of at least one frame per second. A "camera-only" system does not rely on additional sensors such as LIDAR for performing the reconstruction.

	Dense Maps	(Near) Real-time	Outdoor	Object Tracking	Object Dense Reconstruction	Semantic Cues	Large-scale	Camera-only	Active Noise Reduction
Reddy et al. [62]	✓		✓	✓		✓	∼	✓	
Vineet et al. [70]	✓	✓	✓			✓	✓	✓	
KinfuSeg [82]	✓	✓		∼	∼			✓	
Kochanov et al. [36]	✓		✓	✓	∼	✓	✓	✓	
Jiang et al. [33]	✓		✓	✓	✓				
Faeulhammer et al. [16]	✓	∼			✓			✓	
Co-Fusion [46]	✓	✓	∼	✓	✓	✓		✓	
Ours	✓	✓	✓	✓	✓	✓	✓	✓	✓

2.6 Dense SLAM for Dynamic Environments

This section covers several related works which aim to operate in dynamic environments, tracking and, in most cases, also reconstructing the moving objects within.

Table 2.1 shows an overview of the systems which will be presented in the remainder of this section.

In their 2015 paper, Reddy et al. [62] describe a stereo-based SLAM pipeline capable of both building dense maps of the static environment, as well as tracking the dynamic objects within. A key aspect of their system is the fact that it is capable of enforcing global consistency in the map by using bundle adjustment, ICP-based visual odometry, and a series of semantic cues based on TextonBoost. However, the scale of the final reconstructions is relatively limited, with the longest showcased sequence consisting of just 212 frames from a KITTI video sequence, i.e., a little over 21 seconds. The system also does not densely reconstruct the dynamic objects which it tracks. The paper does not provide any details on the run time of the system.

Vineet et al. [70] describe an extension of Nießner et al.'s voxel hashing framework [55] capable of generating semantically labeled maps from stereo input. The system is made robust to potentially dynamic objects, such as cars, by

means of semantic cues; areas of the map which belong to potentially dynamic classes, such as "car" or "bus," are updated more aggressively when new measurements are received, thereby enforcing the overall consistency of the map, preventing dynamic objects from corrupting it. The system is capable of running in near real-time, but does not track or reconstruct the dynamic objects from the environment.

KinfuSeg [82] is an extension of the KinectFusion pipeline which increases its robustness to dynamic objects, separately reconstructing the static and dynamic parts of a scene. The system, however, suffers from the same scale limitations of the original KinectFusion work, being unable to reconstruct environments larger than a single room. Moreover, the system is limited to reconstructing a single dynamic object, being unable to separate multiple dynamic objects based on e.g., their motion or aspect. Finally, being based on an active RGBD sensor, the system is confined to indoor environments, limiting its applicability in the field of autonomous outdoor robotics.

Kochanov et al. [36] use stereo cameras to build a dense semantic map of an environment which also incorporates information on the dynamics of the scene. Their method is centered around augmenting a sparse voxel grid to also include a running estimate of each cell's scene flow, in addition to the semantic and occupancy information. The map updates occur in a Bayesian fashion: at every stage, the occupancy and semantic information from the map is first propagated using a motion model based on the known scene flow. Afterwards, the new measurements (occupancy, scene flow, semantics) are fused into the map, and global consistency is enforced by means of a 3D CRF. The voxel grid is represented using voxel hashing, thus enabling the system to scale to very large scenes. However, the described pipeline does not run in real time. Among its components, it leverages the dense scene flow method of Vogel et al. [71], which can take up to 300 seconds for a single frame. Moreover, by representing the 3D map using an occupancy grid, the system forfeits the many advantages of more elaborate representations, such as TSDF- or surfel-based ones, leading to less smooth, lower-quality reconstructions.

Jiang et al. [33] also present a SLAM system for dynamic environments which reconstructs both the static map, as well as the dynamic objects within. The strength of this system lies in its novel approach to motion segmentation, which performs 3D sparse subspace clustering on the trajectories of keypoints in 3D, grouping together those features which exhibit similar motion, and using region growing to densely segment the full moving objects. Their work is somewhat similar to ours, save for three major differences. First, their system requires LIDAR input, in addition to RGB, limiting its applicability in low-cost robotics. Second, the system does not run in real time, requiring roughly ten seconds to process one frame, albeit using just a single CPU core. Third, the scalability of the system in terms of scene size is limited; the longest sequence presented in their paper consists of just 70 frames, i.e., seven seconds of actual driving time. It is unclear whether the presented pipeline is able to scale to larger environments without vastly increased computational costs.

Ambrus et al. [2] and Faeulhammer et al. [16] focus on RGB-D mapping

and object detection in low-dynamic environments. In this context, a low dynamic environment is one where all changes are expected to occur outside the observer's view, i.e., the system is not robust to witnessing motion, but can recognize rooms upon revisiting them, even if e.g., the layout of the furniture has been changed. The former work simply leverages change detection between room visits to identify objects which have been moved such as chairs or pillows. Its follow-up by Faeulhammer et al. adds another layer to the system, enabling the robot to not just detect (low-)dynamic objects, but to also deliberately plan routes to perform additional scans in order to capture previously unseen sections of the detected objects.

Co-Fusion [46] is another real-time dense mapping system capable of reconstructing both a static background map, as well as the dynamic objects from a scene. The paper presents a real-time pipeline which can segment and track dynamic objects based on either motion or semantic cues, and reconstruct them separately, using a surfel-based representation. While most of the experiments are performed on indoor RGBD sequences, both real-world and synthetic, some limited results on the Virtual KITTI dataset [18] are also presented. However, both the quality and the scale of the reconstructions performed on the Virtual KITTI data are limited, in comparison to the indoor sequences. Moreover, the authors do not address the problem of noise reduction for 3D reconstruction, which is an important consideration for dense mapping of outdoor environments.

Chapter 3

Background

This chapter covers the prerequisites for understanding the components which make up the DynSLAM system. While most notions are presented quite briefly, an effort is made to point the interested reader towards further readings wherever appropriate.

3.1 Visual Odometry

A key aspect of any autonomous robotic platform is its ability to sense its own motion, or *egomotion*. While dedicated solutions for directly sensing egomotion do exist, for example in the shape of *inertial navigation systems*, which leverage GPS, inertial measurement unit (IMU), and wheel sensor information to estimate a robot's motion, they are typically expensive and subject to drift, while also being limited in the range of environments in which they can operate.

Visual odometry [56] is an alternative to pose estimation using sensors such as the INS. Instead of requiring IMU, GPS, or wheel encoder information, visual odometry estimates a robot's motion purely based on its visual input, without requiring prior knowledge about the structure of the scene.

In the present work, the egomotion is estimated using the `libviso2` library by Geiger et al. [21]. As motivated in §2.1, their approach provides a good trade-off between speed and accuracy, as well as robustness to outliers. Moreover, since the method uses sparse scene flow as its input, we can later reuse this computation when analyzing the 3D motion of the objects present in the environment.

`libviso2` estimates the six-degree-of-freedom relative pose of a calibrated stereo camera between two consecutive frames by minimizing the reprojection error of 3D points triangulated from the previous stereo pair onto the current one.

At the heart of the method lies a simple but effective system which computes four-way feature matches between the previous and current left and right frames. Given that these matches correspond to the positions of 3D points at times $t-1$ and t, they represent a sparse version of the scene flow. The

matched sparse scene flow points are then used to minimize the aforementioned error function within a RANSAC framework, which conveys robustness to outliers.

3.1.1 Feature Matching

`libviso2` detects keypoints using simple 5×5 blob and corner detectors. The feature matching is performed by comparing Sobel filter responses centered around the keypoints using the sum of absolute differences as an error metric. This avoids the high computational costs typically associated with more elaborate descriptors such as SIFT [44] and SURF [5], which are also not necessary given the small viewpoint difference between consecutive frames, and the left and right cameras.

In order to compute the sparse scene flow at time t, features are matched between both the current left and right, as well as the left and right frames at time $t-1$. Matching starts in the left frame at time t. Every feature is first matched to the best candidate in the current right frame. Then, the match is itself matched to the best candidate in the previous right frame, which is then matched with the previous left frame, and then back with the current left frame. Features for which the circular matching reaches the same place it started are considered accepted.

The circular matches (sparse scene flow) are then used for egomotion estimation. They are also stored for later use in the 3D object pose estimation component of DynSLAM.

3.1.2 Egomotion Estimation

The egomotion estimation is cast as an optimization problem tasked with reducing the sum of squared projection errors of the 3D points triangulated from the previous frame, projected into the current one. The objective (loss) function therefore takes the shape of

$$L\left(\mathbf{X};\mathbf{r},\mathbf{t}\right)=\sum_{i=1}^{N}\underbrace{\left\|x_i^{(l)}-\pi^{(l)}\left(\mathbf{X}_i;\mathbf{r},\mathbf{t}\right)\right\|^2}_{\substack{\text{Squared reprojection error}\\ \text{into the current } \textbf{left} \text{ frame.}}}+\underbrace{\left\|x_i^{(r)}-\pi^{(r)}\left(\mathbf{X}_i;\mathbf{r},\mathbf{t}\right)\right\|^2}_{\substack{\text{Squared reprojection error}\\ \text{into the current } \textbf{right} \text{ frame.}}}, \tag{3.1}$$

where $\mathbf{X}_i$ represents the 3D position of the ith point triangulated from its two correspondences in the previous frame (out of N total matched keypoints), $x_i^{(l)}$ and $x_i^{(r)}$ are the 2D positions of that same keypoint in the current left and right frames, and $\pi^{(l)}$ and $\pi^{(r)}$ are matrices which project 3D points from the previous frame onto the current left and right image planes, respectively. $\mathbf{r}$ and $\mathbf{t}$ are the rotation and translation components of the six-degree-of-freedom relative pose between two frames. Note that there is no scale ambiguity due to the stereoscopic nature of the input. The optimal transformation $(\mathbf{r}^*,\mathbf{t}^*)$ between the previous and the current frame is then estimated using the Gauss-Newton method to minimize the above objective function:

$$\left(\mathbf{r}^*,\mathbf{t}^*\right)=\underset{\mathbf{r},\mathbf{t}}{\arg\min}\, L\left(\mathbf{X};\mathbf{r},\mathbf{r}\right). \tag{3.2}$$

In order to increase its robustness to outliers, this method is then wrapped in a RANSAC framework which repeats the (fast-converging) optimization procedure for k different samples each consisting of three four-way correspondences. In practice, we set k to 200, as this provides a good trade-off between efficiency and accuracy for our dense fusion method.

The output of this method is the relative pose of the camera between two consecutive frames, i.e., its egomotion. This result is then passed on to the rest of the pipeline, namely the dynamic object tracking, where knowledge of the egomotion is required in order to establish an object's independent motion, and to the static map fusion, which requires the camera's most recent pose in order to properly fuse the latest measurements into the global map.

It is worth noting that `libviso2` is a pure visual odometry library, and does not perform loop closure detection or any global pose graph optimization. Given that the primary focus of this work is the reconstruction of street-sized environments and of the objects within in a single pass, using pure visual odometry for pose estimation is sufficient. For possible ways of adding support for relocalization, loop closure, and globally consistent mapping to DynSLAM, please see Chapter 6.

3.2 Depth from Stereo

3.2.1 Overview

The task of estimating the depth of every pixel seen by a camera in a scene, using nothing other than the camera's color input is one of the fundamental problems of computer vision. While this problem has been studied widely in various forms, using single-, two-, or even multiple-view geometry, for the scope of this work we will be focusing on the two-view case, also known as stereoscopic (or stereo) vision.

The stereoscopic depth estimation problem consists in computing a depth map for a pair of images by associating a depth value to every pixel in the two images. In this scenario, as opposed to structure from motion, for instance, it is assumed that the stereo rig is calibrated, i.e, both the intrinsic matrices of the two cameras, as well as their relative pose are known.

Depth cannot be unambiguously extracted from a single traditional camera. This is because of the fundamental limitations of projective geometry. As such, a monocular camera can be seen as a "bearing sensor", measuring not the distances to various features in a scene, but their angles with respect to the camera's optical axis.

However, viewing a scene from more than one camera, as is the case in the stereo vision problem, imposes an additional constraint on the problem, allowing the depth of a 3D point to be recovered, as a function of its disparity: the distance between the point's projection in the left and in the right camera frames.

Figure 3.1 shows a simplified example of depth from stereo, with identical cameras which are aligned with the x-axis. This enables the depth Z_p of a

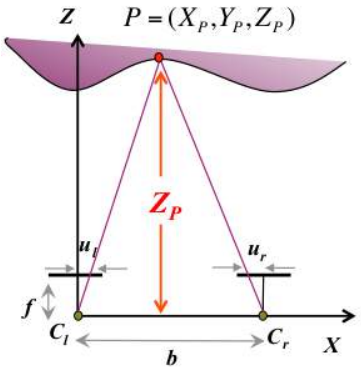


Figure 3.1: A simplified geometric example for estimating a pixel's depth in the stereo depth scenario. Illustration credit: Prof. Margarita Chli, *Autonomous Mobile Robots* Lecture Slides, ETHZ, 2016.

pixel p to be estimated as

$$Z_p = \frac{bf}{u_l - u_r}, \tag{3.3}$$

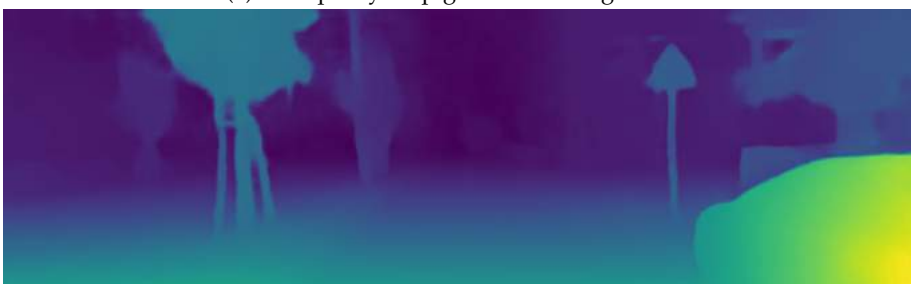
where b is the baseline of the stereo rig, i.e., the distance between the camera centers on the x-axis, f, the focal length of two (assumed identical) cameras, and $u_{l,r}$ the u-coordinates of a 3D point in the left and right images.

Therefore, knowing the cameras' intrinsic parameters, their relative pose, as well as the disparity value of every pixel means that a depth map of the viewed scene can be extracted. However, this raises two further questions: (1) How are the disparity values computed? and (2) Can this computation be done efficiently?

The first question can have multiple different answers depending on the desired properties of the resulting depth map. For instance, matching robust feature descriptors such as SIFT [44] between the two frames can lead to very accurate results, but can only produce sparse depth maps, which are insufficient in dense mapping, for example. The alternative is to attempt to match *every pixel* in one image to a pixel in the other image by, e.g., comparing their surrounding image patches using a simple error metric such as normalized cross-correlation. This technique is commonly referred to as *block matching*.

While the latter approach can lead to reasonably accurate depth maps which are much denser than those produced by matching sparse keypoints, the computational costs of finding a match for every pixel in one image can become overwhelming, growing quadratically as a function of the input resolution, since in the general case, the best match for a given patch in the left image can be located anywhere in the right one. This provides a segue into the answer for the second of the aforementioned questions. The efficiency of the simple block matching technique can be improved substantially by reducing the size of the required search area for every disparity. This is performed through a process known as *stereo rectification*, which transforms a pair of images such that the epipolar lines are all parallel and collinear. Stereo (or

(a) A disparity map generated using ELAS.



(b) A disparity map generated using DispNet.

Figure 3.2: A comparison of the disparity maps produced by ELAS and DispNet. Note that qualitatively, the map produced by ELAS tends to be sharper, while the one produced by DispNet boasts 100% completeness but is substantially more blurry. The gaps in the former do not pose a significant issue when fusing information over multiple frames.

epipolar) rectification typically also removes various types of lens distortion before applying the final affine correction. After the rectification, the block matching search in the right frame can be performed efficiently along the same scan line as the original pixel in the left frame.

Nevertheless this technique still assumes a very simplistic geometric model of the scene, and while efficient, it can still lead to noise and gaps in the resulting depth maps.

There are numerous ways of improving this naive solution, as described in Chapter 2. As motivated in that chapter, both ELAS [20] and DispNet [47] will be evaluated in this work. A comparison of the disparity maps produced by these two methods can be seen in Figure 3.2.

3.2.2 Efficient Large-scale Stereo (ELAS)

ELAS is a disparity estimation technique developed by Geiger et al. [20] whose source code is openly available as `libelas`[1]. It can be seen as a hybrid depth estimation method, leveraging both sparse and dense matching in order to produce its final result.

[1] http://www.cvlibs.net/software/libelas/

This method solves much of the ambiguities typically associated with stereo matching by first computing a high-quality sparse "support" depth map by robustly matching keypoints between the two images. It then computes the Delaunay triangulation of the support keypoints, using it to guide the computation of the dense result.

Concretely, ELAS formulates the dense depth estimation as a Bayesian inference problem, with the prior proportional to a combination of a uniform distribution and a Gaussian distribution parameterized by the distance between the output point and the support points. The likelihood term is expressed as a Laplace distribution parameterized by the ℓ_1-distance between the descriptors of two image patches.

The resulting disparity map is then computed by performing maximum a posteriori inference (MAP) in the aforementioned model.

The method runs on a modern CPU in roughly 130ms, or at approximately 8Hz, and can be run in parallel to other components, such as the visual odometry and the semantic segmentation.

3.2.3 DispNet

Unlike many traditional approaches to disparity estimation such as semi-global matching, the DispNet method proposed by Mayer et al. [47] uses an end-to-end neural network architecture to directly estimate full-resolution disparity maps from rectified stereo input pairs.

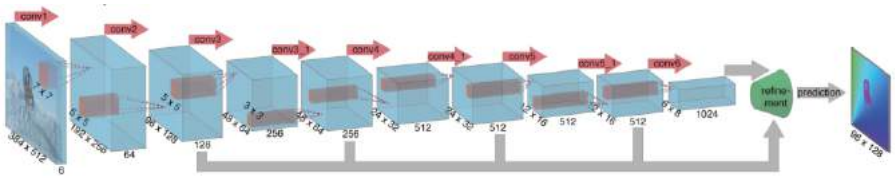
DispNet has an encoder-decoder structure, with additional long-range connections between the encoder and the decoder.

The authors present two alternate configurations for depth estimation: one with an explicit correlation layer, and one without it. In the former version, the first part of the encoder is bifurcated, and has separate branches for the left and right frames, while in the latter, both frames are fed together into the network. The reasoning behind the former version is to encourage the network to produce meaningful representations of both the left and the right frame before focusing on the disparity estimation, while the reasoning for the later is simplicity. The two architectures are presented in Figure 3.3. In our system, we use the first version, as it performs better than the second one on the KITTI 2015 stereo benchmark [49].

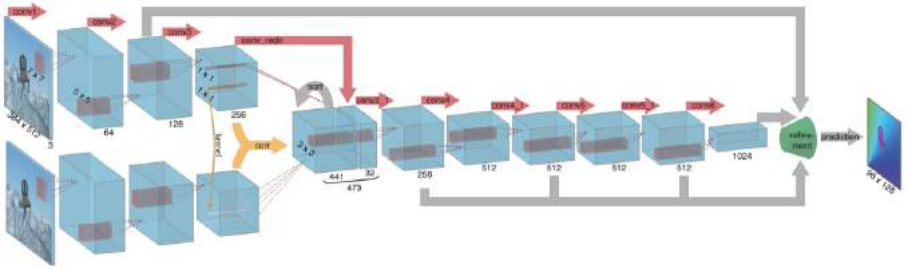
The network is trained end-to-end on a novel synthetic dataset also introduced in the same paper, and fine-tuned on the training dataset of the KITTI 2015 stereo benchmark.

When compared to `libelas` as in Figure 3.2, DispNet produces denser, but less sharp results. It is capable of estimating depth even in the presence of occlusions, as can be seen behind the tree to the left and the sign to the right, but it is also susceptible to "hallucinating" geometry in places where information is scarce, such as the top left portion of the tree.

While computing disparity maps with DispNet is substantially faster than with ELAS (18Hz vs. 8Hz), doing so requires a GPU, which means that depth



(a) The simple version of DispNet. The input images are stacked to form a (width × hight × 6)-dimensional input.



(b) The correlation-based version of DispNet. The input images are fed separately to two branches of the encoder as two (width × height × 3)-dimensional inputs, and their representations are only fused after four layers.

Figure 3.3: The two DispNet architectures for end-to-end disparity estimation from stereo pairs. The *refinement* funnel is a placeholder for the decoder half of the network. Note that this particular example shows optical flow as the output, but the architecture for estimating disparity is essentially identical. Illustration from Dosovitskiy et al. [14].

map computations with DispNet cannot happen in parallel to the object detection described later in this chapter. As will be detailed later, this can constitute a rather large disadvantage in terms of overall pipeline performance, assuming a single-GPU system.

For additional background on deep learning, including architecture types, optimization, and additional applications, we would like to refer the reader to [25].

3.3 Dense Mapping

3.3.1 Relation to Simultaneous Localization and Mapping

The simultaneous localization and mapping (SLAM) problem is concerned with enabling an autonomous agent, such as a robot, to localize itself in an unknown environment while also constructing a map of it, without leveraging any prior knowledge.

The first formulation of the modern simultaneous localization and mapping (SLAM) problem dates back to Smith et al. [67], who in 1986 first showed how increasing correlations between observed landmarks can help reduce

Figure 3.4: An augmented reality vehicle simulation using a dense map created by the Dense Tracking and Mapping system [54] to achieve realistic physics. This highlights one of the many possible applications of dense SLAM systems.

the uncertainty about a robot's location, and lead to a globally consistent, reliable map.

In their comprehensive 2006 tutorial, Bailey et al. [4] provide a detailed overview of the SLAM problem and the main ways of approaching it, focusing on "classic," sparse methods.

While traditional SLAM methods represent maps as sets of sparse keypoints in 3D or keyframes, many modern SLAM systems choose to instead represent the maps as dense 3D models. Naturally, this requires much more computational power and storage, but can lead to both improved localization performance, as well as richer maps, which can be leveraged for higher-level tasks such as simulations, path planning, augmented reality, etc.

A more recent survey from 2015 by Cadena et al. [8] compares many of the newer, dense methods to the classic methods of the past, highlighting that despite their higher computational costs, dense and direct methods are generally more robust to, e.g., featureless environments, with their output maps being more appropriate for applications such as visualization, simulations, path planning, etc.

The potential of dense maps was first highlighted by one of the earliest real-time dense SLAM systems, Dense Tracking and Mapping (DTAM) [54], which included a realistic physics simulation based on the dense map produced by the system. An example of augmented reality based on this method can be seen in Figure 3.4, where a user can control the 3D vehicle on top of the desk, using the dense reconstruction of the environment to enable realistic collision detection and physics.

The release of the Microsoft Kinect SDK in Spring 2011[2], together with the increasing flexibility and availability of general-purpose GPU (GP-GPU) devices and frameworks, sparked a host of new research involving dense methods based on color and depth cameras, commonly referred to as RGB-D cameras.

One of the most influential such works is Newcombe et al.'s KinectFusion [53], which has paved the way for dozens of papers leveraging its volumetric map representation, as well as its techniques for applying projective data association and the iterative closest point (ICP) algorithm for frame-to-model camera tracking. Similar to DTAM, this system also made heavy use of GP-GPU, and was able to run in real time.

While much of the work on classic SLAM also covers the global consistency problem, often using methods such as bundle adjustment to jointly optimize the entire map and all available camera poses, many dense formulations [53, 55, 38, 70] rely exclusively on frame-to-model tracking or pure visual odometry for pose estimation. In our work we do the same, leveraging visual odometry for simple frame-to-frame tracking and focusing on locally-accurate maps and robust dynamic object reconstructions. The task of enforcing global map consistency is left as future work (see §6.2).

As motivated in Chapter 2, our reconstruction components are based on InfiniTAM [38], which extends the original KinectFusion framework by adding support for voxel block hashing, significantly improving its scalability to large scenes. For the remainder of this section, we will focus on InfiniTAM, describing the KinectFusion model it is based on, and presenting further details about voxel block hashing.

3.3.2 The KinectFusion Model

Based on earlier work by Curless and Levoy [11], the volumetric fusion model used by KinectFusion has been at the heart of many subsequent works thanks to the flexibility and robustness of its tracking, data association, and volumetric fusion components. We will focus on the latter two since they are the most relevant to our work, given that we rely on a sparse method for the camera tracking, as described in §3.1.

One of the key insights of the KinectFusion model is its map representation, which is based on a *truncated signed distance function* (TSDF). As illustrated in Figure 3.5, the signed distance function encodes the distance to the closest represented surface at every point in space, with positive values being associated with the exterior of a surface, and negative ones with its interior. This is known as an *implicit surface representation*, as it implicitly encodes the location of a surface at its zero level set. The size of the discretized cells dictate the resolution of the reconstruction.

Formally, a signed distance function is a function

$$S : \mathbb{R}^3 \to \mathbb{R}, \quad S(\mathbf{p}) = \begin{cases} -d, & p \in \text{Volume} \\ d, & p \notin \text{Volume} \end{cases}, \tag{3.4}$$

[2] https://www.microsoft.com/en-us/research/blog/mixing-it-up-the-kinect-for-windows-sdk/

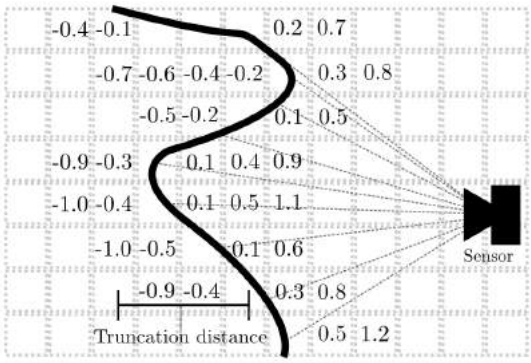


Figure 3.5: A visualization of a discretized truncated signed distance function (TSDF) representing a 2D line implicitly as its zero-crossing. The value of the function is positive outside the represented shape, and negative inside it, with the absolute value being equal to the distance to the nearest point of the surface. Illustration from Whelan et al. [76].

which maps points $\mathbf{p}$ in three-dimensional space to their distance d from the reconstructed surface.

The truncation limits the maximum value of the encoded distance d, such that the SDF is only defined in a finite band around the true surface, $|d| \leq \mu$. The parameter μ is generally selected as a function of the expected noise magnitude.

There are several advantages to using a TSDF for three-dimensional reconstruction over a classic occupancy grid. First, the TSDF allows the encoded surface to be reconstructed easily via raycasting or marching cubes, without having to seek a mode of a probability distribution. Second, it allows new measurements to be integrated more robustly, by essentially functioning as a "running average" of multiple partially-overlapping observations of the object being reconstructed. This allows the final reconstruction to be refined over time, reducing the impact of sensor noise on its quality. And finally, due to its regular structure, the TSDF representation lends itself well to parallelization, and is therefore highly amenable to GPU processing, as described in §3.5.

The TSDF representation is updated with the new depth information at every frame as follows:

- First, the new camera pose is computed, either using a frame-to-model tracker, like in the original KinectFusion paper, or with a tracker based on sparse features, like in DynSLAM (see §3.1 for further details).
- Each pixel in the new depth image is mapped to one or more TSDF cells TSDF_d with associated weights W_d, which are typically constant, or set as an inverse function of the depth.
- The new measurements are then fused into the main volume as follows: The weight and TSDF values of every voxel $\mathbf{p}$ which has a corresponding value in the new measurement are updated in parallel using a run-

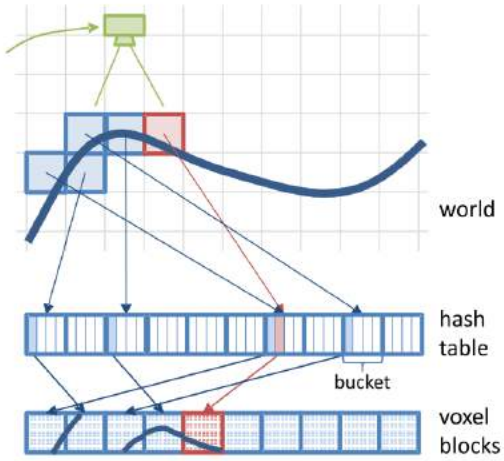


Figure 3.6: The voxel hashing data structure. Voxel block coordinates are hashed and used to look up the corresponding entry which, if present, contains a pointer to the appropriate allocated voxel block in the voxel block array (VBA). Illustration from Nießner et al. [55].

ning average:

$$\mathrm{TSDF}_t(\mathbf{p}) = \frac{W_{t-1}(\mathbf{p})\,\mathrm{TSDF}_{t-1}(\mathbf{p}) + W_d(\mathbf{p})\,\mathrm{TSDF}_d(\mathbf{p})}{W_{t-1}(\mathbf{p}) + W_d(\mathbf{p})}, \tag{3.5}$$

where $W_{t-1}(\mathbf{p})$ represents the weight currently associated with the TSDF cell, and $W_d(\mathbf{p})$ the weight associated with the new depth measurement. Weights are simply accumulated over time, amounting to increased confidence in voxels which are observed over many frames:

$$W_t(\mathbf{p}) = W_{t-1}(\mathbf{p}) + W_d(\mathbf{p}). \tag{3.6}$$

3.3.3 Voxel Block Hashing

KinectFusion operates on dense TSDF volumes, which can be quite wasteful since in nearly all practical cases, most voxels of a scene correspond to empty space outside the truncation band, and do not contribute to the reconstruction. However, due to its inherent structure, the dense volume is unable to avoid storing these uninformative voxels.

First proposed in the context of dense reconstruction by Nießner et al. [55], voxel block hashing alleviates this issue by partitioning the reconstruction into voxel blocks, typically $8 \times 8 \times 8$ voxels in size, which are stored in a hash table using their location as a key, as illustrated in Figure 3.6. This allows the system to only allocate and process blocks which are known to contain meaningful information, i.e., blocks containing voxels located inside the truncation band of the reconstructed surface. Voxel block hashing therefore leads to superior scalability, while retaining the performance and reconstruction accuracy of the original KinectFusion model.

The voxel block hash map is accessed as follows: For a given voxel, the location of its corresponding block is computed by finding the largest multiple of eight smaller than each of its coordinates. The voxel block coordinates are then hashed to an integer key and looked up in the table, iterating over a bucket's elements in the (rare) event of a collision. If the block is found, its corresponding table entry will contain a pointer to the allocated block data.

The InfiniTAM engine from Kähler et al. [38] builds on Nießner et al.'s voxel hashing scheme, bringing several speed and accuracy improvements to their framework. InfiniTAM differs from Nießner et al.'s implementation in the following major ways:

- Unlike the original voxel hashing implementation, InfiniTAM does not make use of any locks when allocating data in the hash table. This is motivated by the fact that collisions are extremely rare during allocation, while also being easy to recover from. This is because it is very likely that a block whose allocation failed at time t will still be visible at time $t + 1$, so its reallocation will be attempted again automatically.
- The raycasting component used in voxel block allocation, tracking, fusion, and visualization is optimized by having an initial step which pre-computes the visible blocks before performing the main voxel-wise rendering.
- The system maintains a list of currently visible blocks, significantly reducing the computational costs of the integration and rendering stages. Newly allocated blocks are always added to the visible list, and the list is pruned at every frame, removing blocks which are no longer visible. While the computational costs associated with maintaining it are negligible, the benefits are considerable, since it allows the system to only account for visible blocks when fusing information or rendering the most recent viewport. This can substantially reduce computational costs in large scenes.
- In addition to CUDA, InfiniTAM also supports hardware acceleration using Apple Metal.

3.4 Instance-aware Object Segmentation

Object detection is the task of determining the locations and approximate dimensions of the objects present in a scene, typically in the form of axis-aligned bounding boxes. Applications of this problem include augmented reality, sports analytics, surveillance, as well as autonomous robotics. The task of object detection can also be extended to *object segmentation*, which is tasked with locating the precise outlines of the objects present in a scene.

In the most general case, objects are not restricted to a particular class, and are detected generically based on a set of "objectness" criteria. Alexe et al. [1] state that an object must satisfy at least one of the following criteria:

- It must have a well-defined, closed boundary.
- It must be different from its surroundings.

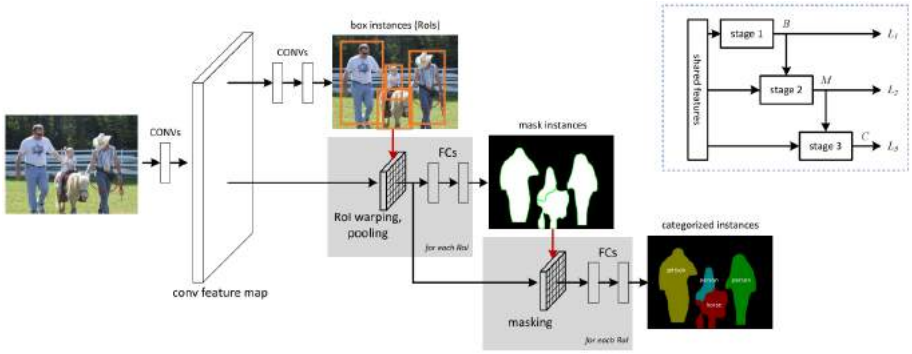


Figure 3.7: An overview of the Multi-task Network Cascade architecture from [13].

- It *may* be unique in an image.

However, in DynSLAM knowledge of the semantic class to which a detected segment belongs is also necessary. This is due to the fact that we are only interested in objects with the potential to exhibit dynamic motion in common urban environments, e.g., we are interested in pedestrians and cars, but not dining tables or chairs.

To this end, we leverage the Multi-task Network Cascades (MNC) architecture developed by Dai et al. [13], which not only localizes and segments object instances, but also outputs a semantic label for every detection. They achieve this by using a three-layer cascade of sub-networks for which they develop an efficient training scheme. The cascade is organized as follows:

1. The first network produces object proposals using a variant of the Region Proposal Networks used in the Faster R-CNN system [63]. These proposals take the form of simple bounding boxes, which are refined in the following stages.
2. The second network processes the bounding boxes produced by the previous stage together with the existing image features, outputting a pixel-wise binary instance mask.
3. The third network takes the instance masks, bounding boxes, and image features computed before and assigns the detection one of $N + 1$ classes, taking all pixels in the window into consideration but giving more weight to those present in the mask. Here, N is the number of objects classes, with the extra class representing the background, i.e., a proposal identified as a false positive.

Figure 3.7 presents a graphical overview of the cascade architecture.

3.5 General-Purpose GPU Programming

The notion of *General Purpose Computations using the Graphics Processing Unit*, commonly abbreviated as *GP-GPU*, refers to using a computer's dedicated

graphics processing unit (GPU) for performing computations that are traditionally handled by the CPU [17].

A GPU typically encompasses several streaming multiprocessors (SMP), each consisting of hundreds or even thousands of dedicated cores. While the complexity of the individual cores is much smaller than that of a standard (e.g., x86) CPU core, the strength of a GPU comes from the sheer number of cores it contains; at the time of writing, a high-end GPU such as a Titan Xp^{TM} consists of over 3800 cores, two orders of magnitude more than a traditional CPU. Modern GPUs also have dedicated high-bandwidth memory, separate from the CPU-accessible RAM.

This specialized architecture can lead to massive speed-ups for data-parallel computations, such as raytracing or matrix multiplications. A common example of the impact of GP-GPU has had on the computer science research community is the recent success of deep neural networks [37], which was made possible by leveraging GPUs to accelerate the computationally expensive matrix multiplications associated with both training and inference. Both DispNet [47], one of the approaches we use for disparity estimation, as well as the Multi-task Network Cascades [13], leveraged for performing semantic object detection make use of GPU acceleration for efficient training and inference.

The availability of GP-GPU has also contributed to the growth of dense mapping systems, traditionally considered too computationally intensive for real-time applications. The original KinectFusion pipeline [53] made heavy use of GP-GPU in both its tracking, and its mapping components, in order to achieve real-time operation capabilities. DynSLAM also makes heavy use of the GPU, both directly, for the primary mapping tasks, as well as indirectly, in its disparity estimation and semantic segmentation components.

CUDA is a GP-GPU framework created and maintained by the nVIDIA Corporation. Due to the wide availability of nVIDIA GPUs, as well as to the performance and flexibility of the framework itself, CUDA has become the *de facto* standard of the GP-GPU world, and is the dominant force in fields such as dense mapping and deep learning.

In the CUDA framework, the GPU device[3] is controlled from a CPU program which executes operations for GPU memory allocations, memory transfers between host (CPU) memory and device (GPU) memory, as well as code execution.

The GPU-specific code takes the form of C-style functions called *kernels*, whose execution is controlled by the host program, and which are run in parallel by the GPU.

The basic workflow associated with CUDA programming is typically as follows (the associated CUDA API function names, which follow the same patterns as the C standard library, are also presented for reference):

1. Allocate memory on the GPU (`cudaMalloc`).

[3]We will assume single-GPU systems for simplicity, but CUDA also supports multi-GPU configurations.

2. Copy the input from RAM to the GPU memory (`cudaMemcpy`).
3. Execute the kernel, specifying the number of threads it should run on.
4. Copy the results of the kernel operation back to RAM for visualization, further processing, evaluation, storage, etc. (`cudaMemcpy`).
5. Free the GPU memory (`cudaFree`).

For example, inputs in DynSLAM can take the form of computed depth maps for the volumetric fusion, or color images for the semantic segmentation. Data-parallel operations such as residual calculations, depth fusion, and ray-tracing take the form of GPU kernels, and are invoked by the host code at every frame as needed. Finally, outputs such as reconstruction previews and exported meshes must be copied from GPU memory back to RAM, before they can be post-processed, evaluated, saved to the disk, etc.

While the InfiniTAM system on which our work is based also supports the Apple Metal framework, as well as plain CPU operation, we chose to focus on CUDA when implementing our system. This is both for performance and practical (availability of CUDA-ready GPUs) reasons. Moreover, both the semantic segmentation component of our system, as well as one of the disparity estimation components, DispNet, are based on the Caffe framework, which only supports CUDA-based acceleration at the time of writing[4].

[4]While an OpenCL (https://www.khronos.org/opencl/) branch of Caffe does exist, it is experimental, and incompatible with DispNet and the Multi-task Network Cascades implementations, which are both based on forks of the official CUDA version of Caffe.

Chapter 4

Dynamic Reconstruction

4.1 Overview

The primary goal of DynSLAM is to reconstruct all potentially dynamic rigid objects encountered in a scene, in addition to the static background map. This ensures that the map remains uncorrupted and suitable for, e.g., relocalization, frame-to-model tracking, visualization, etc., while the generated dense object models can be leveraged to improve tracking and motion prediction, and to perform higher-level tasks such as path planning.

The current work focuses on generating high-fidelity reconstructions of both static maps and dynamic objects on long input sequences in a variety of environments, leaving higher-level applications for future work.

We choose to leverage semantics instead of motion as our primary cue for detecting objects as this allows us to recognize and reconstruct not just moving objects, but also static ones which have the potential to transition to being dynamic while being observed, such as parked cars. We use sparse scene flow to compute the detected objects' 3D motion which is then compared to the camera's egomotion in order to determine whether a particular object is moving independently. Knowledge of this 3D motion is also required when reconstructing moving objects, in order to correctly fuse their views from multiple frames. This process is described in more detail in §4.3.

For the scope of this work, we focus on rigid object tracking and reconstruction, particularly cars, but the system can be extended to support the reconstruction of objects belonging to other, possibly non-rigid, classes, such as pedestrians and cyclists. For more details, please see §6.2 in Chapter 6.

The system is implemented in C++, leveraging CUDA for performing parallel GPU programming where appropriate, like for the map regularization described in §4.5. The source code is available online, at https://github.com/AndreiBarsan/DynSLAM.

The remainder of this chapter is structured as follows: We begin by briefly describing each pipeline component, in order to give the reader a sense of how everything fits together, and what the primary challenges are. In the sections following that, we delve deeper into the novel components of the pipeline, ex-

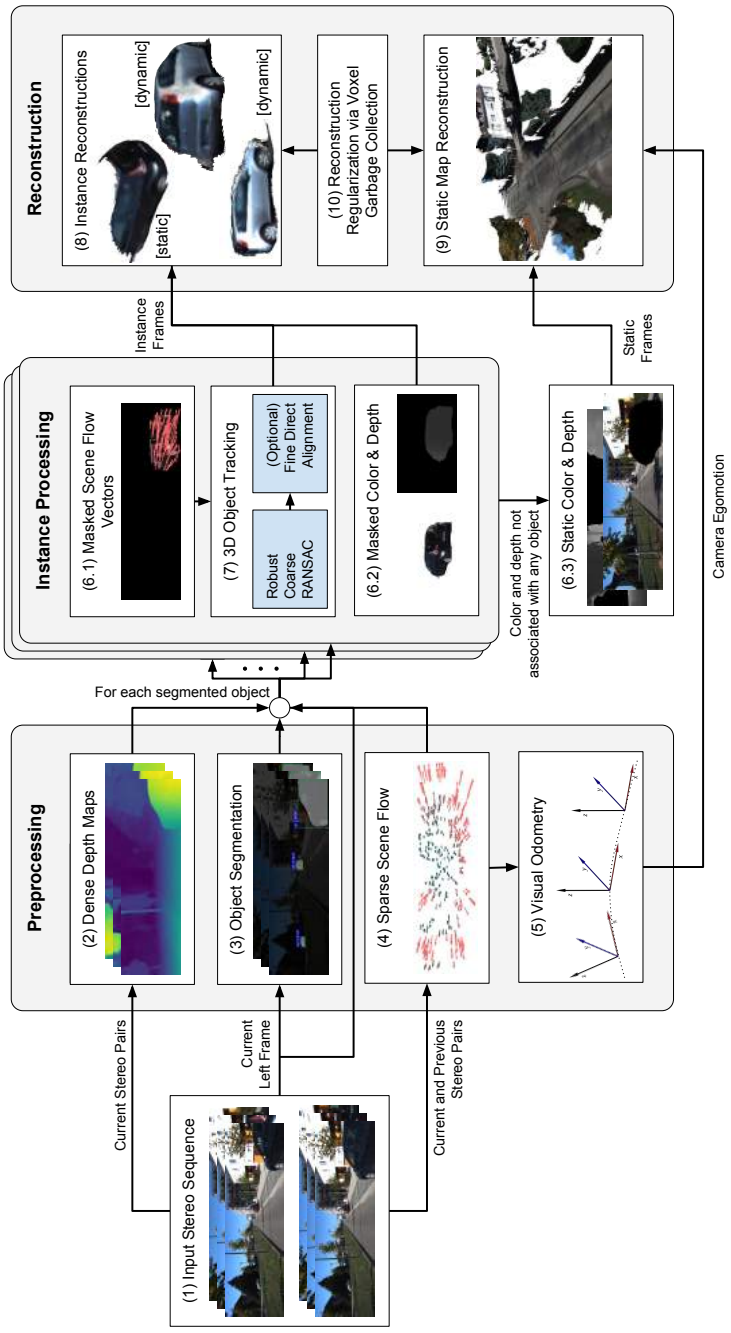


Figure 4.1: An overview of DynSLAM's pipeline.

plaining them in more detail, highlighting the design decisions which were made, and discussing the strengths and weaknesses of our approach.

As shown in Figure 4.1, the main components of DynSLAM are the following:

(1) Input The pipeline reads rectified stereo image pairs as input, such as those from the well-known KITTI Vision Benchmark [19]. No GPS, IMU, or LIDAR information is required.

(2) Dense Depth Maps The dense depth maps in our pipeline can be computed using either Efficient Large-scale Stereo Matching (ELAS) [20], or the DispNet neural network architecture [47]. The two methods are described in detail in §3.2. Their impact on the reconstruction quality is compared thoroughly in Chapter 5.

(3.1) Object Segmentation The semantic object segmentation is performed using the Multi-task Network Cascades architecture from [13]. This component detects and classifies object instances in an input image, using the 20 labels from the Pascal VOC2012 dataset. The details of this method are presented in §3.4.

(3.2) 2D Object Tracking The object segmentation component operates on individual frames. It therefore has no concept of object identity across frames. Because of this, our pipeline must associate every detection in a new frame to a track, creating new tracks where appropriate, and accounting for possible gaps in the tracks due to, e.g., detection failure. We perform this task using a technique which associates new detections with existing tracks by ranking them based on the Intersection-over-Union score between a new detection and the most recent frame in a track. Further details are provided in §4.2.

(4) Sparse Scene Flow Following the method described by Geiger et al. [21], the computation of the sparse scene flow is based on two-view and temporal stereo. Namely, simple blob and corner features are matched between the current left and right frames, and the previous ones, resulting in pairs of 3D points from consecutive time steps $t-1$ and t, i.e., the scene flow.

The sparse scene flow is used both for egomotion estimation in Step (5), as well as for computing the 3D motion of the objects detected in a scene, in Step (7). The details of this method are covered in §3.1.

(5) Visual Odometry The vehicle egomotion is computed from the sparse scene flow using `libviso2` (Geiger et al. [21]), which relies on a RANSAC-based approach to robustly compute visual odometry in environments with large proportions of outliers, such as dynamic street scenes. As with the sparse scene flow, the details of this method are covered in §3.1.

(6.1) Instance-specific Scene Flow The scene flow vectors corresponding to a specific object instance can be computed by simply masking the full-frame scene flow computed in Step (4) with the detected object's silhouette.

(6.2) Instance-specific Color and Depth Similar to the sparse scene flow, the color and depth frames are also masked using the relevant object silhouettes, yielding *instance views*. Instance views are "virtual input frames" with all the information outside an object instance removed. When the instance views are created, the corresponding segments are also removed from the original input frames.

(6.3) Static Color and Depth The color and depth parts of the input frame not associated with any (potentially) dynamic object are considered part of the *static view* and are fused into the static map in Step (9). Figure 4.3 shows detailed examples of static and instance views.

(7) 3D Object Tracking This component estimates each tracked object's six-degree-of-freedom 3D motion, which it then compares to the egomotion computed in Step (5) in order to classify objects as static or dynamic.

A coarse estimate of an object's 3D motion can be computed from its corresponding scene flow vectors. This part of this process is very similar to the one used for the visual odometry in Step (5). If successful, this coarse motion estimate can optionally be used to bootstap a finer method based on direct image alignment. Both of these techniques are described in §4.3.

(8) Individual Object Reconstructions As described in §3.3, DynSLAM uses InfiniTAM[1] [38] to perform volumetric fusion. Each object reconstruction is computed in a separate InfiniTAM volume, using its corresponding instance view as input at every frame. For objects detected as static, this process is identical to the reconstruction of the static map, relying on the camera's egomotion alone to align the frames, since the object itself has no independent motion. For dynamic objects, the estimate computed in Step (7) is used.

(9) Static Map Reconstruction The static views from Step (6.3) are aligned using the visual odometry computed in Step (5) and fused into the static map. Just like the object reconstruction, the static map reconstruction is also performed using InfiniTAM. The processes for reconstructing the static map and the objects are described in detail in §4.4.

(10) Map Regularization The volumetric fusion results can exhibit unwanted streak-like artifacts, as highlighted in Figure 4.2. This is due to the inherent noise associated with the estimation of depth from stereo, as detailed in [22, 41]. The artifacts lead to decreased map accuracy and increased memory usage. We address this issue using a specialized regularization method, which we describe in §4.5.

4.2 2D Object Tracking

Given that the object segmentation component presented in §3.4 has no concept of inter-frame identity, operating on individual frames, the 2D detections must be post-processed in order to organize them into tracks. This then

[1] http://www.robots.ox.ac.uk/~victor/infinitam/

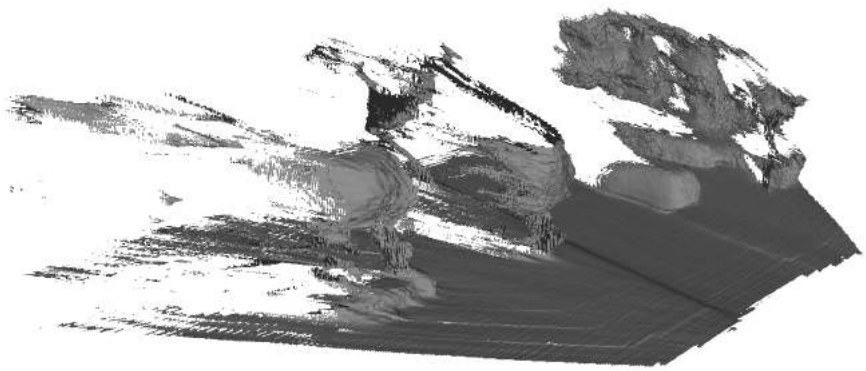


Figure 4.2: An example of the streak-like artifacts produced in 3D reconstructions by noisy depth maps. This example uses four fused depth frames computed by DispNet, with the maximum depth truncated at 20 meters. Note the trailing halos behind the two trees on the left side of the image.

allows us to estimate each object's 3D motion, and to use it to register its measurements in the same coordinate frame for accurate volumetric fusion.

Due to the high accuracy of the detected object segments in each frame, as well as the fact that we are not interested in tracking distant objects which are impossible to reconstruct anyway, we found that a simple approach based on Jaccard similarity (or Intersection-over-Union) works very well for our purposes. Namely, we use the following formulation for our similarity metric:

$$J(M_1, M_2) = \frac{|M_1 \cap M_2|}{|M_1 \cup M_2|}, \tag{4.1}$$

where M_1 and M_2 are the masks of the two segments to be compared.

The 2D track construction proceeds as follows:

- For every new segment detected at time t, find candidate tracks by intersecting the segment's 2D bounding box with the track's most recent bounding box. Tracks with the most recent frame at a time less than $t - 1$ are discounted appropriately.
- We then remove candidate (frame, track) pairs whose score is below a fixed threshold and greedily associate each frame to its highest-scoring track, ensuring no frame is assigned to more than one track.

We found that for our purposes, matching frames to tracks based on the bounding box alone was accurate enough. This is also much faster than performing pixel-wise mask intersections at every frame. In the future, more elaborate matching techniques such as the MAP multi-frame data association formulation from [81] could be used to improve the 2D tracking quality.

Figure 4.3: The static component of the background is separated from the object instance color and depth frames. On top, we see a sample color frame where a potentially-dynamic and a dynamic vehicle were extracted from the static background while below, we see the corresponding extracted frame for the dynamic vehicle.

4.3 3D Object Tracking

After the 2D tracking concludes, the next step consists in attempting to estimate the 3D motion of each object between the previous and the current frame. This is performed using a coarse-to-fine scheme starting with a fast sparse RANSAC-based method followed by a finer-grained direct alignment, initialized using the result of the sparse step, if available.

We use the term **instance view** to refer to a masked version of an input frame only containing a particular object's color, depth, and scene flow information. When processing object detections with classes that have the potential to be dynamic, we extract their silhouettes from the input scene flow, depth, and color frames into new instance views. Their data in the original input is removed, to ensure that dynamic objects do not corrupt the final static map.

Figure 4.3 shows examples of the color component from the static part of the input, as well as from one of the instance views extracted from it. Note that the segmentation is not perfect, and there are still small parts of the background visible in the car's frame and vice-versa. However, this is not an issue, as the volumetric reconstruction is robust to small imprecisions, while larger ones can still be dealt with effectively in the regularization process which will be described in §4.5.

For every object instance which has the potential to be dynamic, e.g., a car but not a dining table, the system attempts to estimate its motion relative to the camera using a method similar to the visual odometry from [21] which is described in §3.1. The input to this coarse motion estimation is the sparse

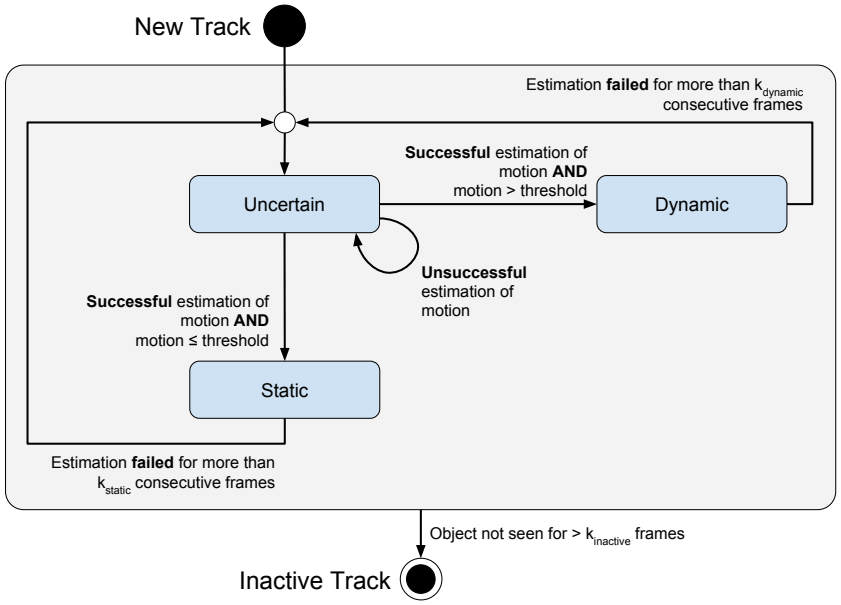


Figure 4.4: A state diagram for the active tracks in the DynSLAM system.

scene flow contained in the instance view, and originally computed as part of the visual odometry estimation.

This works by treating the 3D object as static, and attempting to estimate the motion of a virtual camera around it. If its computation is successful, then the resulting virtual camera motion is equal to the inverse of the object's own motion, expressed in world coordinates.

DynSLAM can label active object tracks as uncertain, dynamic, or static depending on the result of their 3D motion estimation. If the coarse estimation of an object's motion is unsuccessful, such as when the object is too distant for its motion to be accurately estimated, it is flagged as *uncertain*. If the motion estimation is successful, then the system compensates for the previously computed camera egomotion and evaluates the magnitude of the vehicle's motion. If it is greater in terms of either rotation or translation than a particular threshold, then the object is flagged as *dynamic*; otherwise, it is labeled as *static*. Figure 4.4 provides a visual overview of these states, and the transitions between them. For the purpose of static/dynamic classification, an object's 3D motion is compared to the egomotion using the odometry comparison metrics described in [19]. This involves using the magnitude of the translation and the angle of its rotation for the classification.

k_{dynamic} and k_{static} are thresholds indicating the maximum number of consecutive frames during which relative motion estimation is allowed to fail, before a dynamic or static track "falls back" to an uncertain state. In practice, we set them to 2 and 3, respectively, as we found that larger values can lead to corrupted volumetric reconstructions due to the buildup of uncertainty in the object's position. For short gaps in tracks where motion can not be estimated,

using a constant velocity assumption has been found to work well.

If the motion estimation is successful and the object is flagged as dynamic, then the pipeline attempts to refine the object motion estimate using a dense alignment scheme similar to the one used to compute visual odometry in [42].

4.3.1 Coarse Alignment using RANSAC

In order to compute the motion of an object, we first leverage the sparse scene flow computed in the first part of the pipeline, which is masked using the current segmentation silhouette.

As described in the previous subsection, the masked scene flow associated with a specific object instance is used as input to estimate the motion of a virtual camera with respect to the object instance, which is assumed to be static. If the estimation is successful, then the 3D motion of the object is equal to the inverse of the virtual camera's motion. For static objects, this obviously means that the resulting 3D object motion will be nearly identical to the camera's egomotion. This can be used to classify objects with known motion as static or dynamic.

If the object motion at the previous frame is known, then we use it to initialize the estimation process, which can improve its convergence rate.

The estimation process is based on RANSAC, and uses the same formulation as the visual odometry, which optimizes the 6-DoF relative pose using the Gauss-Newton method on a nonlinear least-squares objective penalizing the reprojection error of the features from the previous time step into both the left and right frames of the current one. The process is described in more detail in §3.1.2.

Table 4.1: The `libviso2` parameters used for the RANSAC-based pose estimation.

Parameter	**Value**
Matching: n_{nms}	3
Matching: half resolution	false
Matching: multi-stage	true
Matching: refinement	per-pixel
Maximum RANSAC iteration:	200
RANSAC inlier threshold	2.0
Maximum features per bucket	15

Table 4.1 shows the `libviso2` parameters used in the sparse scene flow and object instance motion estimation procedures.

It is worth noting that we do not need to perform such masking when computing the camera's egomotion, since the employed method is designed to be robust to outliers. In other words, we always use *all* the available scene flow vectors when computing visual odometry. We have investigated the effect

of only computing visual odometry on the flow vectors associated with the static parts of the scene, but we did not see any improvements in accuracy or convergence rate.

4.3.2 Pose Refinement Using Direct Image Alignment

After the initial motion estimate is computed using RANSAC, the system can attempt to refine it further using a semi-dense alignment procedure which minimizes the sum of squared photometric errors for all high-gradient pixels. This approach is based on the work of Liu et al. [42].

The optimization objective is the sum of squared photometric errors over $\Omega\left(\mathbf{I}^t\right)$, the set of high-gradient pixels present in the current instance view:

$$\mathbf{E}\left(\mathbf{T}^{t,t-1}\right) = \sum_{i \in \Omega(\mathbf{I}^t)} r_i^2, \tag{4.2}$$

where r_i is the ith photometric residual,

$$r_i = \mathbf{I}^t(\mathbf{u}_i) - \mathbf{I}^{t-1}\left(\boldsymbol{\pi}\left(\mathbf{T}^{t,t-1}\boldsymbol{\pi}^{-1}(\mathbf{u}_i, d_{\mathbf{u}_i})\right)\right). \tag{4.3}$$

The optimization is performed with respect to $\mathbf{T}^{t,t-1}$, the transformation from the previous to the current frame, which is initialized using the output of the coarse method. $\mathbf{I}^t$ and $\mathbf{I}^{t-1}$ are the intensity images at times t and $t-1$, $\mathbf{u}_i$, the coordinates of the ith pixel, and $\boldsymbol{\pi}$, the projection function mapping homogeneous 3D points in the camera's coordinate frame to dehomogenized 2D pixel coordinates. Similarly, the function $\boldsymbol{\pi}^{-1}$ maps a 2D pixel location $\mathbf{u}_i$ and its corresponding depth value $d_{\mathbf{u}_i}$ to a 3D point expressed in homogeneous coordinates.

Just like in the coarse case, the direct photometric alignment is computed by iteratively minimizing the above error function using the Gauss-Newton method.

Using this additional step incurs additional overhead while providing only very small improvements in reconstruction quality, so we chose to disable it for most experiments, except those where its impact on 3D object tracking quality is compared against the RANSAC-only version, in §5.3.

4.4 Volumetric Fusion in Dynamic Environments

The volumetric fusion is the central component of the DynSLAM pipeline. It is tasked with fusing color and depth information from every frame into a common representation in an accurate and robust manner. We use the convention that the left color camera's frame is the canonical coordinate frame of the car, in which everything else, such as the depth maps, the visual odometry, and the volumetric fusion are expressed.

We reconstruct both the dynamic objects and the static map using the open-source InfiniTAM framework [38] as follows:

- For every track which just became eligible for reconstruction, initialize its InfiniTAM instance.

- For every new frame belonging to a static or dynamic track, remove its silhouette from the input color and depth frames, and place it into an object instance frame.
- For every new frame belonging to an uncertain track, remove its silhouette from the input color and depth frames, preemptively.
- Perform fusion and regularization for every object instance being reconstructed.
- Perform the static map fusion and regularization.

As soon as the relative pose of an object instance between at least two frames is known, its reconstruction can begin, starting with the frame before the first successful relative pose estimation.

Once an object instance becomes eligible for reconstruction, the system initializes its own voxel space, as a new instance of InfiniTAM. The origin of this new coordinate system corresponds to the camera pose at the time of the first fused instance frame. The parameters for the reconstruction are similar to those used by the static map, with the obvious exception of the maximum number of voxel blocks, which is set dynamically, based on the configured voxel size, in order to accommodate a $5m \times 10m \times 5m$ volume, which is sufficient for most cars and vans encountered in the KITTI dataset. If this upper threshold is reached, no more voxel blocks are allocated, but existing ones can still be updated based on new measurements.

It is worth emphasizing that the system starts reconstructing all possibly-dynamic objects as soon as the first relative pose is known, irrespective of whether the objects are moving independently or static. This is a key aspect of the system, and the motivation behind it is to allow DynSLAM to support cases where objects transition from being static to dynamic, such as when a parked car starts moving, or when cars which were first detected waiting for at a traffic light depart as the light turns green.

This constitutes a significant advantage over systems which detect objects based on motion cues alone, since it ensures DynSLAM is aware of potentially dynamic objects *before* they start moving, which is very advantageous for autonomous driving in crowded urban environments.

Before a potentially dynamic object can be confidently labeled as static or dynamic, it is flagged as uncertain. This happens when the object's motion cannot be estimated accurately enough, such as when it is far away from the camera (more than 20 meters, typically), and its mask does not span enough useful scene flow vectors to allow for an accurate estimation of the relative pose between two frames of the object's track. Objects flagged as uncertain are not reconstructed, since their frames cannot be registered in a common coordinate frame due to the lack of relative pose information, but they are still preemptively removed from the color and depth frames which are fed to the static map component, in order to avoid corrupting the map on the off chance that the object is, in fact, moving independently, i.e., dynamic. Further details about the possible states of an active track are presented in §4.3.

Finally, after the object instance reconstruction takes place, the resulting color and depth buffers from which all potentially dynamic objects were removed

are fed to the static mapping system.

While the InfiniTAM framework does include a series of frame-to-model trackers, such as an ICP-based one and a direct image alignment one, they are not robust enough for outdoor operation under rapid camera motion, making them unsuitable for our application. Instead, as described earlier in this chapter, we rely on the robust sparse visual odometry from [21] for computing the egomotion, and the method described in §4.3 for determining the motion of other moving objects.

Once a track has been inactive for more than a set number of frames (in practice, we use the relatively aggressive number of 3), its reconstruction is saved to disk and deallocated in order to save GPU memory. This works well in the sequences on which we evaluated our system, but is not robust to longer occlusions. In the future, we plan on using a more sophisticated occlusion-aware tracking mechanism. This would require storing the vehicle reconstructions in memory for longer periods of time, which can be achieved using the voxel block swapping techniques described in [70, 55], which allow voxel blocks to be moved to RAM when they become inactive, and back into GPU memory once they are needed again.

4.5 Map Regularization

While performing volumetric reconstruction at a small scale, such as when scanning room-sized environments using an RGBD camera, the magnitude of the noise associated with the depth sensor is relatively small, compared to that associated with estimating depth from stereo.

The active nature of most RGBD sensors, such as the Kinect or the ASUS Xtion can lead to relatively low noise levels in their output stream, with pixels whose depth could not be estimated being left blank.

RGBD sensors are, however, not without their downsides. As discussed in Chapter 3, their range is very limited, and is typically less than ten meters[2]. This makes them unsuitable for outdoor robotics in large environments, where stereo cameras are preferred. Nevertheless, estimating depth information from stereo cameras comes with its own set of challenges.

Figure 4.2 shows an example of the artifacts produced when rendering a stereo depth map in 3D. It can be seen that the outlines of most objects have a comet-like trail facing away from the direction from which they were perceived. This effect is typically not present in reconstructions using the shorter-range RGBD sensors.

The nature of these "streaks" makes them stretch far behind the object being reconstructed, causing visual artifacts which also lead to the spurious allocation of large numbers of voxel blocks, increasing the memory footprint of the maps, and reducing the system's overall scalability.

In order to reduce the effect of this noise, and improve reconstruction quality while reducing the memory consumption of our system, we turn to a simple

[2]The Kinect v2 sensor has a maximum (reliable) depth range of 4.5 meters, and the ASUS XtionPro™ has a range of 3.5 meters [32]

but effective technique first proposed by Nießner et al. [55]. In their work, they describe a garbage collection method for removing blocks allocated due to noise and moving objects. For every voxel block in the reconstruction, they compute the minimum absolute value of the TSDF and the maximum weight. If the maximum weight of a voxel block is zero, or the minimum value of the TSDF is below a fixed threshold, the block is deleted, freeing up its slot in the hash table and adding its VBA slot to the free list.

We extend this method in two ways. First, we increase its granularity, making it capable of operating on a per-voxel basis, allowing it to "delete" voxels considered noisy even when the block that they belong to does not itself get deleted. Second, we improve this method's scalability by allowing it to take advantage of InfiniTAM's visible block lists, preventing it from having to run on the entire map at every frame.

Note that in the scope of this thesis, we use the terms "map pruning," "map regularization," and "voxel garbage collection" interchangeably.

Voxel Deletion The first way we enhance the voxel garbage collection from Nießner et al. is by enabling it to also operate on individual voxels, ensuring that low-weight stray voxels always get cleared. This is performed purely for aesthetic reasons, as memory savings can only be achieved when operating on a per-block basis. A voxel is cleared by resetting its color and SDF values to black and the maximum SDF value, i.e., their default values.

The conditions for a voxel's deletion are the following:

- It must belong to a block whose age (time since allocation) is smaller than a fixed threshold, k_{minAge}. This ensures that newly allocated blocks do not get collected before they have a chance to accumulate more measurement.
- It must have a (depth) measurement weight smaller than a threshold k_{weight}, *or* a signed distance function (SDF) value larger than a threshold k_{TSDF}. The former condition ensures that the voxel was observed more than a fixed number of times, while the latter allows voxel blocks which are "far enough" from the actual surface to be flagged for deletion.

After the individual voxels are processed, the non-empty voxels in every block are counted using a simple block-level parallel reduction. Blocks found to consist entirely out of empty voxels are deleted from the hash table. Their corresponding voxel memory block is also released and re-added to the free block list.

Algorithm 1 shows a high-level overview of the proposed voxel garbage collection method. The required parameters are:

- t, the current time, i.e., frame number. This is used for accessing the appropriate visible block list.
- k_{TSDF}, the maximum TSDF value for a legitimate voxel. Voxels with a TSDF value above this get deleted.
- k_{weight}, the weight below which voxels are eligible for deletion.

Algorithm 1 Map Regularization through Voxel Garbage Collection

```
1: procedure MapRegularization(t, k_TSDF, k_weight, k_minAge, hashTable, visible)
2:     for all block ∈ visible_{t−k_minAge} in parallel do
3:         if age(block) > k_minAge then
4:             for all voxel ∈ block in parallel do
5:                 if TSDF(voxel) < k_TSDF ∨ weight(voxel) < k_weight then
6:                     voxel ← ∅
7:             usedVoxelCount ← CountNonEmpty(block)
8:             if usedVoxelCount = 0 then
9:                 lockBucket(computeBlockHash(block))
10:                freeBlock(block)
11:                deleteEntry(hashTable, block)
12:                unlockBucket(computeBlockHash(block))
```

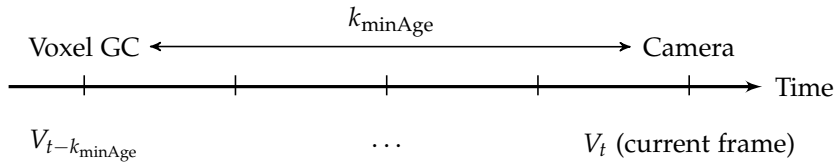


Figure 4.5: Timeline of the voxel garbage collection system. V_i represents the list of visible blocks at time i.

- k_{minAge}, the minimum block age for a voxel to be eligible for deletion. This is still necessary, despite only operating on visible block lists from `minAge` frames ago, because certain blocks from `minAge` frames ago could still be visible at the current time t. This can happen in one of the following scenarios:
  - The camera is stationary or moving very slowly.
  - The camera has performed a (small) loop and is now viewing the area seen approximately minAge frames ago.
- `hashTable`, the GPU-backed hash table used for accessing the voxel blocks.
- `visible`, a list of visible block lists corresponding to every frame.

Scalability One of the main limitations of the original voxel garbage collection implementation is the fact that it always runs on every allocated voxel block. This means that every collection operation processes a large number of blocks which have not changed since the previous collection, leading to substantial redundancy. As our second improvement to the method, we address this issue.

As described in Chapter 3, InfiniTAM keeps track of a list of visible blocks for the most recent frame, for efficiency reasons. By storing this list at every time step, we can record a history of which blocks were visible over time. This list can be leveraged to significantly improve the efficiency of the voxel garbage

collection as follows: Knowing that voxels belonging to blocks younger than k_{minAge} should never be deleted, and that voxels belonging to blocks older than that have already been processed and are very likely unchanged, given the rarity of loops in driving scenarios, at every time t we only consider the blocks which were visible at time $t - k_{\text{minAge}}$ for collection. In other words, DynSLAM performs voxel garbage collection in lockstep with fusion. After processing the nth input frame, allocating the corresponding voxel blocks, fusing the depth map, accounting for dynamic objects, etc., the system performs voxel garbage collection on the blocks visible at the time of input frame $n - k_{\text{minAge}}$. Figure 4.5 shows a simplified timeline of this process.

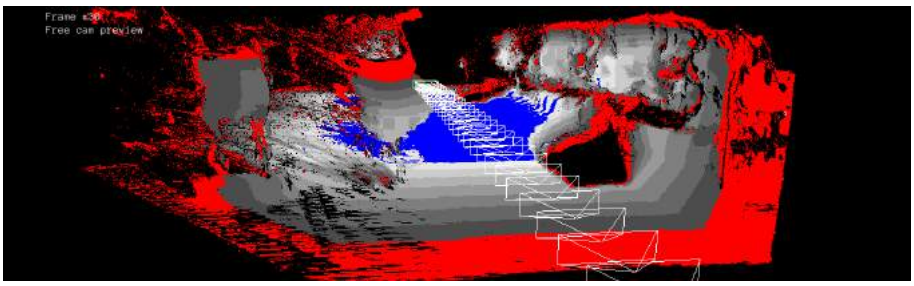
In scenes where the static map can easily occupy hundreds of thousands of blocks, the visible list at every given frame typically holds around 8,000–12,000 blocks, resulting in a 10-fold reduction of computational costs, as compared to running the process over the entire volume at every step. On an nVIDIA Titan X, this typically results in an overhead of less than 1ms at every frame. Detailed results in terms of memory savings and map accuracy improvements are presented in §5.2.

While this process is necessary for large maps, when reconstructing vehicles we can afford to simply run the collection process for the entire volume, at every frame, given their much smaller scale. Moreover, once a dynamic object leaves view, we perform one final, more aggressive voxel garbage collection (with a higher k_{weight}) before (optionally) saving its model to the disk.

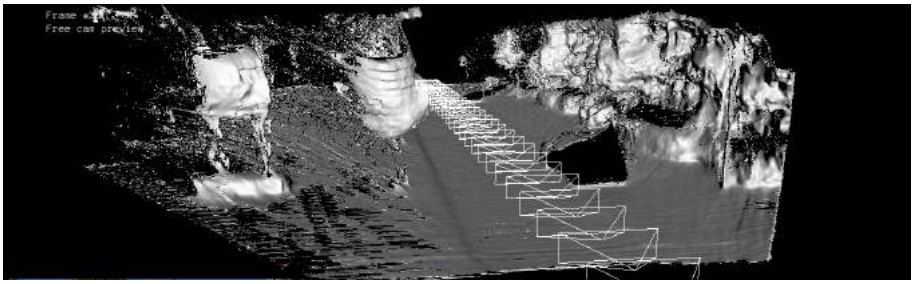
Weight Visualization In order to explore the voxel weight distribution in a map, we added a new rendering mode to the InfiniTAM engine, enabling the visualization of individual voxels' weights in real-time. A preview of this mode is shown in Figure 4.6. It can be seen that much of the noise associated with the streaking artifacts is rendered in red, signifying that it is eligible for garbage collection. The thick red band at the bottom of the reconstruction corresponds to the start of the sequence, and was only seen in one frame, leading to the low weight associated with it.

Implementation Details Implementing the voxel garbage collection in the InfiniTAM framework required a number of changes to be made to the core of the volumetric fusion engine.

The GPU hash table implementation from InfiniTAM is lock-free, which means it is very high-performance, but that it does not support deletions, which would require locking a bucket to ensure that the consistency of the hash table and the excess list are maintained upon item deletion. Moreover, the original InfiniTAM engine makes heavy use of absolute indices to the hash table and/or the excess list. The typical use case replaces the usage of keys with raw table indices, in order to avoid additional hashing operations and hash table look-ups. An example of this is the original visible list. Instead of containing the keys, i.e., coordinates of the visible blocks, it stored raw offsets into the hash table for performance reasons. However, once the hash table is required to support deletions, storing raw indices is no longer an option, as deletions can cause them to become stale, leading to corruption.



(a) Voxel weight visualization in InfiniTAM.



(b) A shaded version of the same view.

Figure 4.6: Visualizing voxel weights in InfiniTAM. The red voxels are eligible for deletion. The gray voxels' intensity is proportional to their weight. Blue voxels are saturated voxels whose measurement weight is equal to the maximum weight.

To this end, we made two major modifications to the core InfiniTAM engine: First, we updated all the code which relied on absolute indices, such as the block allocation code, and the visible block list, to correctly address elements in the hash table by their key, instead of raw indices. Second, we added support for the deletion operation to the hash table, ensuring that the buckets are locked when deletions are performed, as described in [55].

While these modifications do lead to a slight decrease in performance for the volumetric fusion, adding roughly 1–2ms of additional overhead per frame, the cost penalty is negligible when compared to other, more expensive components from our pipeline, such as the semantic segmentation or the depth map computation.

The modified version of InfiniTAM used for this thesis as a component of DynSLAM can be found online at https://github.com/AndreiBarsan/InfiniTAM.

Chapter 5

Experimental Results

We evaluate the reconstruction, 3D tracking performance, and memory footprint of our system both quantitatively and qualitatively on a major autonomous driving dataset, the KITTI Vision Benchmark Suite [19]. The dataset encompasses a wide variety of outdoor video sequences recorded from a moving vehicle in both urban and rural areas around the city of Karlsruhe, in Germany. All sequences include a video component, which consists of 1392×512 stereo pairs recorded with a baseline of 0.54m, as well as Velodyne LIDAR point clouds. The dataset also contains INS-based ground truth poses for every frame, as well as ground truth depth, optical flow, scene flow, 2D/3D object tracks, and semantic and road segmentation for selected frames and brief sub-sequences. The sampling rate of the dataset is 10Hz.

The dataset is challenging for multiple reasons. First, the speed of the vehicle varies significantly within sequences, and can be very high in, e.g., freeway sections. Second, the lighting conditions also vary both between sequences, since the dataset covers sunny and cloudy weather, but also within them, as the camera passes through forests and tunnels. Third, there are many specular and transparent surfaces present, such as cars and windows, which can prove to be particularly challenging to algorithms estimating depth from stereo pairs. Finally, other moving objects in scenes, such as cars, bikes, pedestrians, trains, etc., can also move very rapidly, and occlude large parts of the cameras' field of view. Examples from all of these categories are shown in Figure 5.1.

Concretely, we base our experiments on the video sequences from the KITTI odometry and tracking benchmarks [19], using the LIDAR as a ground truth for evaluating the quality of both the reconstructed static maps, as well as the dynamic object instances. We also use the 3D tracking ground truth to evaluate the accuracy of the pose estimation for the dynamic objects. Overall, we evaluate our system on 21 input sequences consisting of over 25000 frames, corresponding to roughly 40 minutes of driving through varied urban and rural environments. Figure 5.2 shows a sample reconstruction from KITTI odometry sequence 13 (the bridge sequence) together with several highlights, including the reconstruction of a dynamic vehicle encountered near the start of the sequence.

(a) A bloom artifact caused by reflected sunlight.

(b) A strong difference in light intensity between the tunnel and the outside.

(c) Cars approaching from the opposite direction result in very fast relative motion, which can be very challenging for an object reconstruction pipeline.

Figure 5.1: Three examples of challenging visual situations encountered in the KITTI dataset.

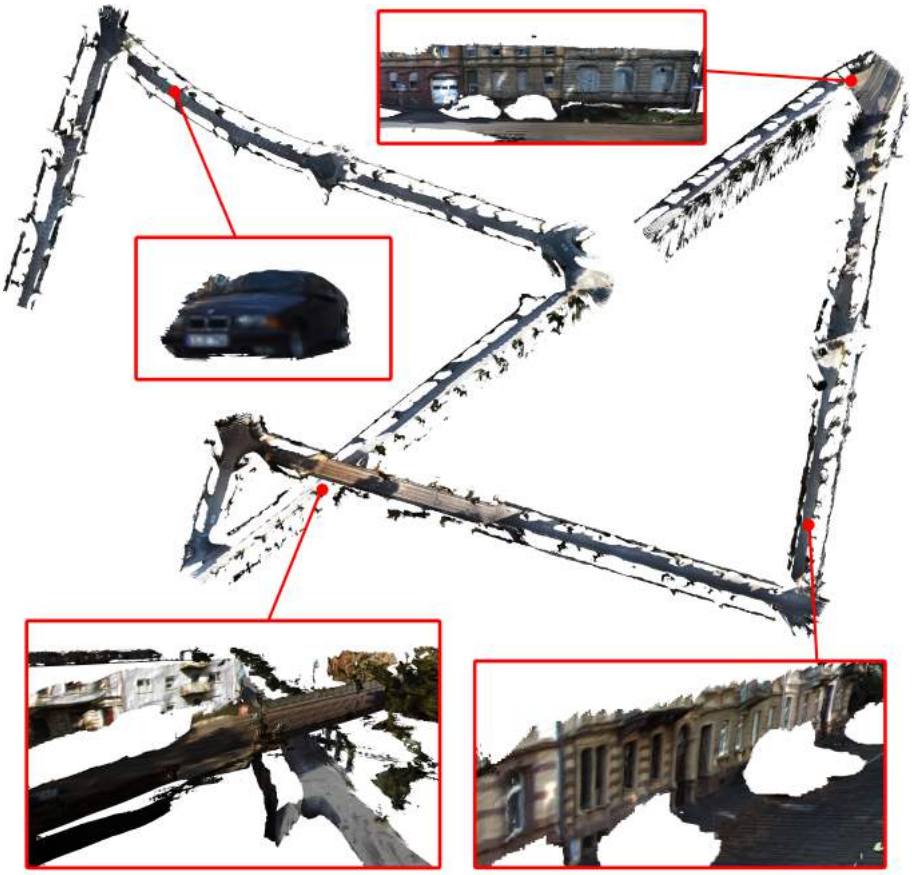


Figure 5.2: The static map created by DynSLAM from the first 1560 frames of KITTI odometry sequence 13. All cars are removed from the map and reconstructed separately. The top-left highlight shows the reconstruction of an independently moving car built on the fly.

Figure 5.3 shows a screenshot of the main user interface for DynSLAM. In addition to allowing all aspects of the pipeline to be visualized, including the segmentation result, LIDAR ground truth, sparse scene flow, etc., it provides tools for controlling the regularization and exporting the reconstructions as meshes. The tool also computes all the accuracy metrics presented in this chapter, offering the option to visualize them in real time.

The remainder of this chapter is structured as follows: first, we present a series of experiments which highlight the reconstruction quality of our system in both static and dynamic scenes, highlighting the benefits of reconstructing the dynamic objects instead of ignoring their information. Next, we quantify the impact of the map regularization introduced in §4.5 on the resulting accuracy, completeness, and memory consumption. Following this, we evaluate the 3D tracking performance of our system, comparing the results of the sparse feature-based pose estimation to the refined results obtained using the direct alignment phase. Finally, we measure the effect of the spatial and

Figure 5.3: The DynSLAM GUI application allows the map and the dynamic objects to be visualized as they are reconstructed. Here, we see a preview of the KITTI-odometry sequence number 5 being reconstructed, around frame 2050. From top-left to bottom-right, it shows (1) a full preview of the active reconstruction together with the camera pose history, (2) a preview of the left camera's input with the LIDAR ground truth superimposed as a sanity check, (3) the reconstructed depth map with all car silhouettes removed, (4) a preview of the frame segmentation result, (5) a preview of the instance-specific color frame for one of the active reconstructions, (6) memory usage statistics, and (7) a novel view of one of the objects being reconstructed separately from the static map.

temporal input resolution on reconstruction quality in a series of ablation studies.

5.1 Reconstruction Quality

5.1.1 Methodology

In order to quantitatively evaluate the reconstruction quality of our system, we compare our generated static map and dynamic object models to the LIDAR point clouds from each frame. This is performed using the method described by Sengupta et al. [65] and Vineet et al. [70]. The method represents a generalization of the standard depth evaluation strategy from the KITTI Stereo Benchmark [49]. It consists in projecting the LIDAR points onto the left camera's plane, and comparing them with the corresponding values of the input and fused frames. The input frames are simply the depth maps computed using ELAS/DispNet at that specific frame, while the fused frames are depth maps synthesized from the active reconstruction, which also incorporates the most recent input depth map.

For measuring accuracy, we only consider LIDAR points which have a corresponding value in both the input depth, and the fused depth. This ensures that the comparison between ELAS and the fused result is fair, even when the input frame is nearly 100% accurate but very sparse. This is not an issue with the DispNet input frames, which always cover the entire input image.

Following the methodology of the KITTI Stereo Benchmark [49], we compare pixel disparities, and consider pixels whose delta disparity is greater than 3px and 5% of the ground truth value as erroneous.

Similarly, we evaluate the completeness of both the input and the fused depth maps by measuring the number of ground truth pixels (i.e., projections of LIDAR points in front of the vehicle) for which a corresponding input/fused depth value exists.

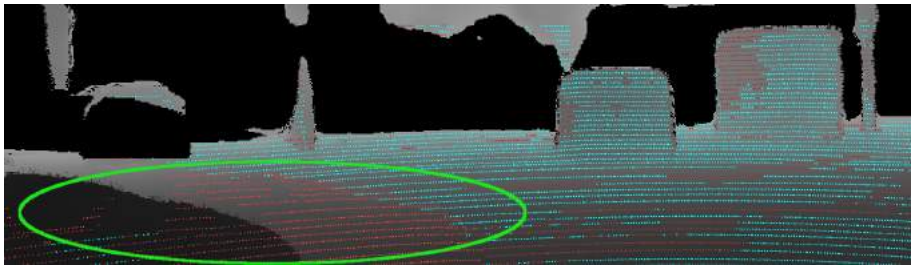
We compute per-frame accuracy and completeness scores and present their averages across the frames in a sequence, together with information about their variance in the shape of box plots.

The box plots presented in, e.g., Figure 5.6, follow the Tukey convention: The bottom and top edges of the boxes correspond to the first and third quartiles of the represented distribution. The bottom and top whiskers stretch up to the lowest and highest data point still within 1.5 IQR (inter-quartile range) of the lower or upper quartile. We have chosen to use box plots because of their ability to effectively detailed information about the underlying distribution, more so than, e.g., simple error bars.

The primary goal of our system, which is the construction of separate dense models of the environment as well as of the objects within, means that simply computing accuracy and completeness scores on full frames is insufficient. The reasons for this are twofold. First, computing full-frame scores does not yield specific information about the different reconstructions built by our system; the results would simply aggregate the accuracy and completeness scores over static and dynamic parts of the environment alike. Second, the full-frame evaluation can mistakenly label correct pixels as erroneous when the static map is occluded by an object our system can track but not reconstruct, such as a biker, or a car which just entered the scene and whose 3D motion is not yet known.

Figure 5.4 shows an example of such a case. In this scenario, the car has just entered the frame, but given that only a small part of it is visible, its 3D motion can not be estimated yet. Therefore, DynSLAM tracks it in 2D, but does not yet begin to do so in 3D, or attempt to reconstruct it (which would require knowledge of its 3D motion). Therefore, the synthesized depth map from DynSLAM's point of view does not yet incorporate this car, as can be seen in Figure 5.4a. Directly comparing this depth map to the input depth map and to the LIDAR ground truth is therefore not appropriate, since it would not account for the detected-but-not-yet-reconstructed car.

In order to solve this issue, we evaluate our system using *semantic-aware evaluation* based on the semantic segmentations computed using the Multi-task Network Cascades. We evaluate the input and synthesized (i.e., fused) depth maps as follows:



(a) An example of misattributed errors caused by evaluating an area of the image specifically not reconstructed as an error. The problematic area is highlighted using the green ellipse.

(b) The voxels evaluated as part of the static map, when using semantic-aware evaluation. The ground truth measurements corresponding to the dynamic object not yet being reconstructed are no longer taken into account.

(c) The original input frame, where the car has just entered view, so its motion is still unknown.

(d) The corresponding instance-aware object segmentation. Everything belonging to the static background is evaluated on its own. Ground truth values belonging to all dynamic objects which DynSLAM has started reconstructing are evaluated together, as a separate metric.

Figure 5.4: Example scenario where aggregating errors over the entire frame introduces bias in the results, by attempting to evaluate an area occupied by a vehicle whose reconstruction was not yet started due to lack of information about its motion.

- Ground truth points associated with no potentially dynamic object are counted towards the *static map* statistics.
- Ground truth points associated with potentially dynamic objects undergoing reconstruction are counted towards the *dynamic object* statistics.
- The remaining ground truth points which correspond to dynamic objects not undergoing reconstruction (e.g., bikes, pedestrians, or distant cars whose 3D motion cannot be computed) are ignored.

This method is, obviously, imperfect, as it relies on the computed semantic segmentation, which is not always fully reliable. Nevertheless, we have found it to work well in practice, allowing us to draw numerous insights about our system's performance under various conditions, as will be described in detail in the following sections. Other possible approaches to this challenging evaluation scenario, such as using simulated data with ground truth 3D models, are discussed in §6.2.5.

While InfiniTAM is already quite robust to outliers and dynamic objects, we show that our method has the potential to further improve the quality of the static maps in challenging scenes, while also robustly reconstructing the encountered dynamic objects.

5.1.2 Experimental Results on the KITTI Odometry Sequences

We present the results on the 11 KITTI odometry training sequences in Figures 5.6–5.9. The experiments lead to several observations.

First, the reconstruction accuracy of the static map is improved slightly by the fusion when using ELAS maps, but not when using DispNet depth maps. This can be explained by the fact that the results of DispNet are noisier and less correlated across frames, reducing the benefits of the fusion.

Second, ELAS leads to more accurate static maps than DispNet (Figure 5.6), but less accurate object reconstructions (Figure 5.7). As illustrated in Figure 5.10, the fact that DispNet is more robust to reflective surfaces and transparency leads to improved performance when reconstructing dynamic objects, i.e., cars, in our case.

Third, the reconstruction accuracy of the dynamic objects is usually not improved by fusion when compared to the input depth maps, and the overall variance in accuracy is much higher. This is to be expected, as vehicles are considerably more challenging to reconstruct than, e.g., road surfaces, fences, and buildings, due to their non-lambertian properties, as well as their independent motion.

Fourth, DispNet leads to much denser (i.e., more complete) reconstructions of both the static map, and of the objects, than ELAS but this gap is reduced by the fusion. This follows from the fact that ELAS depth maps are sparser, but more accurate, meaning that they benefit from the fusion more than DispNet.

Finally, the variance of the ELAS depth map completeness is high because it often produces very sparse results in challenging lighting conditions, as highlighted in Figure 5.5. Nevertheless, the magnitude of this effect is reduced significantly by the fusion process.

(a) A sample input frame with strong shadows and intensity saturation.

(b) The resulting ELAS depth map, which only covers a small proportion of the original image.

(c) DispNet's result is less sharp, failing to reconstruct, e.g., the street light on the right side of the road, but it is also not affected by the lack of texture on the road.

Figure 5.5: A sample input frame where challenging lighting conditions lead to low-coverage ELAS depth maps, without affecting DispNet.

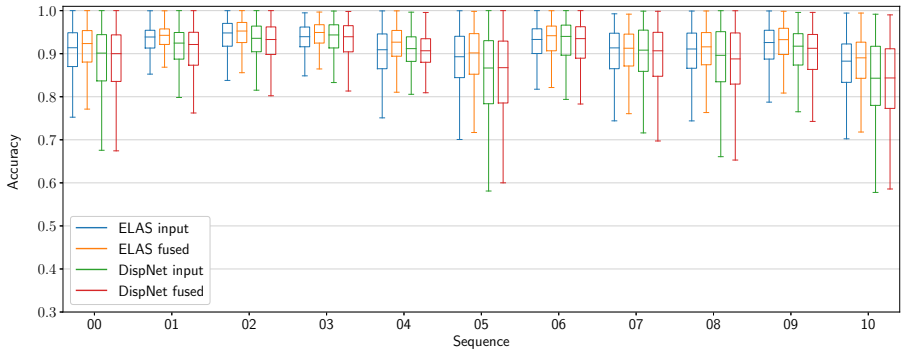


Figure 5.6: Input and reconstruction accuracy on the static parts of the KITTI odometry sequences.

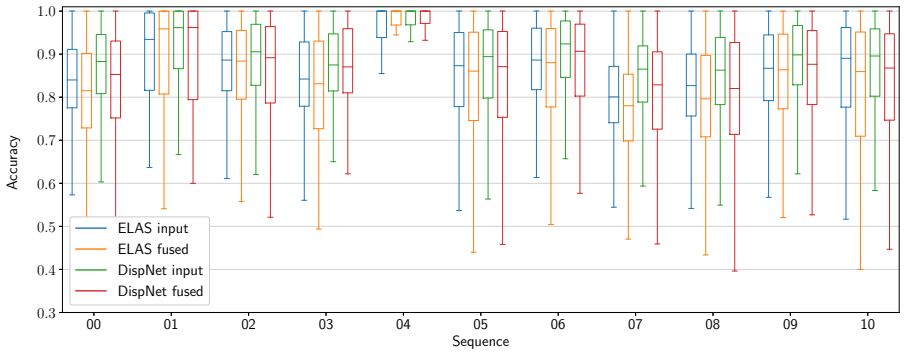


Figure 5.7: Input and reconstruction accuracy on the dynamic parts of the KITTI odometry sequences.

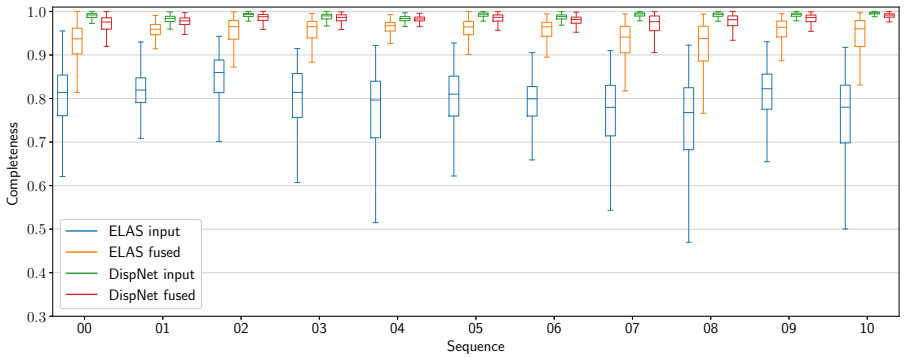


Figure 5.8: Input and reconstruction completeness on the static parts of the KITTI odometry sequences.

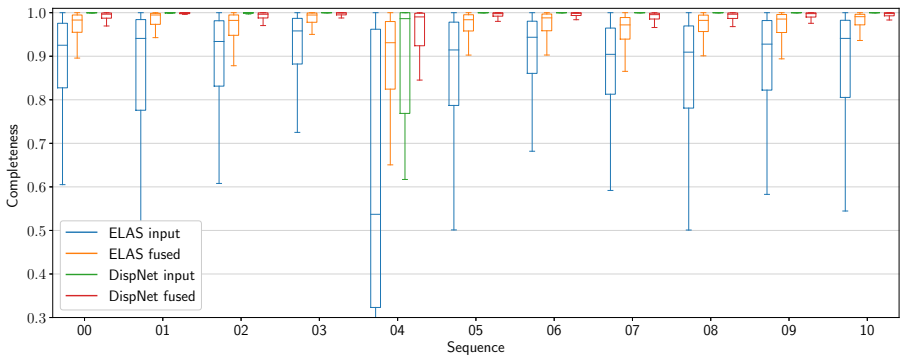


Figure 5.9: Input and reconstruction completeness on the dynamic parts of the KITTI odometry sequences.

Note that the unusual dynamic reconstruction results for sequence 04 reflect its short length (only 271 frames, i.e., 27 seconds of driving), and the fact that almost no object reconstructions actually happen. The sequence features driving on a wide road behind a van which is still too distant to reconstruct, while all other cars approach from the opposite direction, on a distant lane. None of the other cars are visible within the (20m) depth range for more than 2–3 frames, making them impossible to reconstruct reliably, and leading to the deceivingly small error rate and large completeness variance from Figures 5.7 and 5.9.

(a) ELAS depth maps are generally accurate, but unable to deal with reflective or transparent surfaces well, leading to less complete vehicle reconstructions.

(b) DispNet depth maps can enable denser object reconstructions.

Figure 5.10: Two reconstructions of the same car computed by DynSLAM using ELAS and DispNet depth maps. The latter method often leads to more complete results, even under challenging lighting conditions.

5.1.3 Experimental Results on the KITTI Tracking Sequences

Additionally, we use the first 10 training sequences from the KITTI tracking dataset to evaluate the effects of the dynamic object awareness on the quality of the static map reconstruction. To this end, we evaluate the static parts of the input (as described at the beginning of this section), comparing the input depth to the fusion result with and without dynamic object awareness (*Dynamic Fusion* and *Standard Fusion* in Tables 5.1 and 5.2). The former case represents DynSLAM's default mode of operation, which attempts to separate all potentially dynamic objects from the static map, to prevent it from becoming corrupted. The latter is not semantics-aware, essentially operating as an outdoor version of the vanilla InfiniTAM [38] system.

The accuracy and completeness results of this experiment are shown in Tables 5.1 and 5.2, respectively.

As in the previous series of experiments, the fused results outperform the input in terms of reconstruction accuracy on nearly all sequences. Moreover, the dynamic fusion outperforms the standard fusion on most of the sequences. This showcases our method's ability to improve the quality of the static maps by actively preventing dynamic objects from corrupting them.

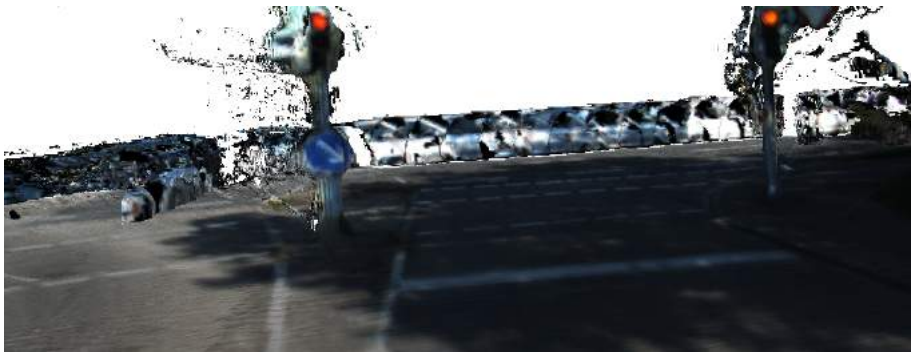
Second, just as observed in §5.1, the reconstructions produced using ELAS are more accurate than those using DispNet, but less complete. This trend can be

Table 5.1: Overview of the static map reconstruction accuracy on the first ten sequences of the KITTI 2012 tracking Benchmark. Bold values indicate the best result within the same depth map category, while stars mark the overall best score on a sequence.

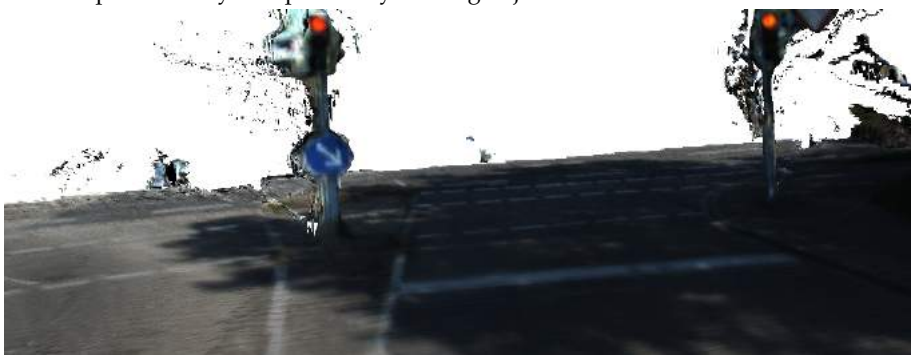
	DispNet			ELAS		
	Input	Standard Fusion	Dynamic Fusion	Input	Standard Fusion	Dynamic Fusion
00	0.8546	0.8402	**0.8743***	0.8512	0.8285	**0.8650**
01	0.9160	**0.9266**	0.9215	0.9218	**0.9373***	0.9316
02	0.9563	0.9635	**0.9635**	0.9677	0.9790	**0.9797***
03	0.8455	0.8522	**0.8591**	0.8875	0.8987	**0.9062***
04	0.8965	**0.9069**	0.9067	0.9117	0.9357	**0.9385***
05	0.8393	0.8354	**0.8400**	0.8990	0.9102	**0.9163***
06	0.9523	0.9667	**0.9707**	0.9602	0.9668	**0.9709***
07	0.8743	0.8825	**0.8838**	0.9009	0.9222	**0.9251***
08	0.9029	0.9086	**0.9123**	0.9267	0.9282	**0.9311***
09	**0.8824**	0.8755	0.8786	0.9184	0.9210	**0.9238***

Table 5.2: Overview of the static map completeness on the first ten sequences of the KITTI 2012 tracking Benchmark. Bold values indicate the best result within the same depth map category, while stars mark the overall best score on a sequence.

	DispNet			ELAS		
	Input	Standard Fusion	Dynamic Fusion	Input	Standard Fusion	Dynamic Fusion
00	0.9847	**0.9869***	0.9773	0.7154	**0.9377**	0.9071
01	0.9835	**0.9839***	0.9646	0.7789	**0.9627**	0.9364
02	0.9856	**0.9896***	0.9872	0.8360	**0.9784**	0.9743
03	0.9761	**0.9805***	0.9742	0.7961	**0.9566**	0.9513
04	0.9883	**0.9888***	0.9837	0.8150	**0.9736**	0.9667
05	0.9828	**0.9857***	0.9822	0.7887	**0.9679**	0.9631
06	0.9886	**0.9921***	0.9902	0.8233	**0.9610**	0.9504
07	**0.9882***	0.9878	0.9733	0.7758	**0.9474**	0.9277
08	0.9853	**0.9853***	0.9820	0.8243	**0.9532**	0.9488
09	**0.9850***	0.9841	0.9734	0.8352	**0.9735**	0.9610



(a) Static fusion is prone to corrupt the environment map with streaks and other artifacts produced by independently moving objects.



(b) Dynamic fusion, the primary operating mode of DynSLAM, prevents vehicle trails and leftover halos from being integrated into the map.

Figure 5.11: Reconstructions produced by Dynamic and Standard fusion on KITTI tracking sequence 01.

seen very clearly in Table 5.1, where ELAS-based reconstructions outperform DispNet-based ones on nearly all sequences. Conversely, Table 5.2 shows the DispNet-based reconstructions outperforming the ELAS-based ones in terms of reconstruction completeness. Interestingly, despite its lower accuracy, standard fusion leads to slightly higher completeness scores than dynamic fusion. This is explained by the fact that object instance removal is not perfect, and tends to sometimes also lead to small areas of the background being removed along with the objects. While this does not affect the quality of the instance reconstructions, with the additional background fragments being prime candidates for voxel garbage collection, it does explain the slightly lower scores of the dynamic fusion in terms of completeness.

At the same time, even in those cases where standard fusion scores better than dynamic fusion in terms of accuracy, such as in sequences 01 and 04, the qualitative results of the latter method still remain superior. An example from sequence 01 is presented in Figure 5.11, where two cars passing in front of the camera leave behind unwanted trails in the static map when dynamic fusion is not enabled. When taking dynamic objects into consideration, the corruption is no longer present. This difference is not reflected in the quan-

titative evaluation because the area containing the corruption is not covered by the ground truth. The evaluation of sequence 04 also exhibits similar behavior.

The dynamic fusion accuracy and completeness scores from sequence 09 are poorer than the input due to the limitations of the segmentation component, which consistently fails to detect a truck driving in front of the camera for several hundreds of frames. This negates all the benefits of dynamic fusion, which cannot succeed if semantic detection fails, leading to the superior scores of the input depth maps, which obviously don't suffer from any fusion-related artifacts. In the future, such scenarios could be avoided by also incorporating motion cues in the object detection component. This is discussed in more detail in Chapter 6.

5.2 Map Regularization

We evaluate the impact that the regularization technique based on voxel garbage collection presented in §4.5 has on DynSLAM's memory consumption and reconstruction accuracy. In order to measure reconstruction accuracy, we use the same metrics described in the previous section. Figure 5.12 shows a top-down comparison of two reconstructions using no regularization and moderate regularization, respectively.

In order to also capture the effect of the voxel garbage collection, which is performed in lockstep with the reconstruction, but with a fixed delay of k_{minAge} frames, we also add a delay to the evaluation. That is, at time t, the voxel garbage collection is processing the blocks visible at time $t - k_{\text{minAge}}$, and the evaluation is performed using the depth map and camera pose from $t - k_{\text{minAge}} - \tau$. The additional offset τ ensures that the map viewed by the camera at that time has been processed by the regularization. In our experiments, we set $\tau = k_{\text{minAge}}$.

Given the limitations of the ground truth, which is provided in the form of per-frame LIDAR readings, we only evaluate the accuracy of the static reconstructions under the effect of voxel garbage collection. The delayed evaluation scheme described above prevents us from also evaluating dynamic object reconstructions, as their position is only known to DynSLAM while they are being observed, and not $k_{\text{minAge}} + \tau$ frames ago.

To this end, we use the same semantic-aware evaluation scheme as in §5.1, with the only difference being that we only evaluate the static map.

We perform our experiments on the first 1000 frames of KITTI odometry sequence number 9, as it contains a small number of dynamic objects, while at the same time being diverse in terms of encountered buildings and vegetation. Even through we also use the regularization for the vehicle reconstructions, they only represent a very small fraction of the system's overall memory usage. We therefore focus on evaluating the memory usage of the static map.

It is also worth noting that due to the nature of most artifacts removed by the regularization, the quantitative evaluations of the subsequent improvement in map quality have a tendency to underestimate the magnitude of the

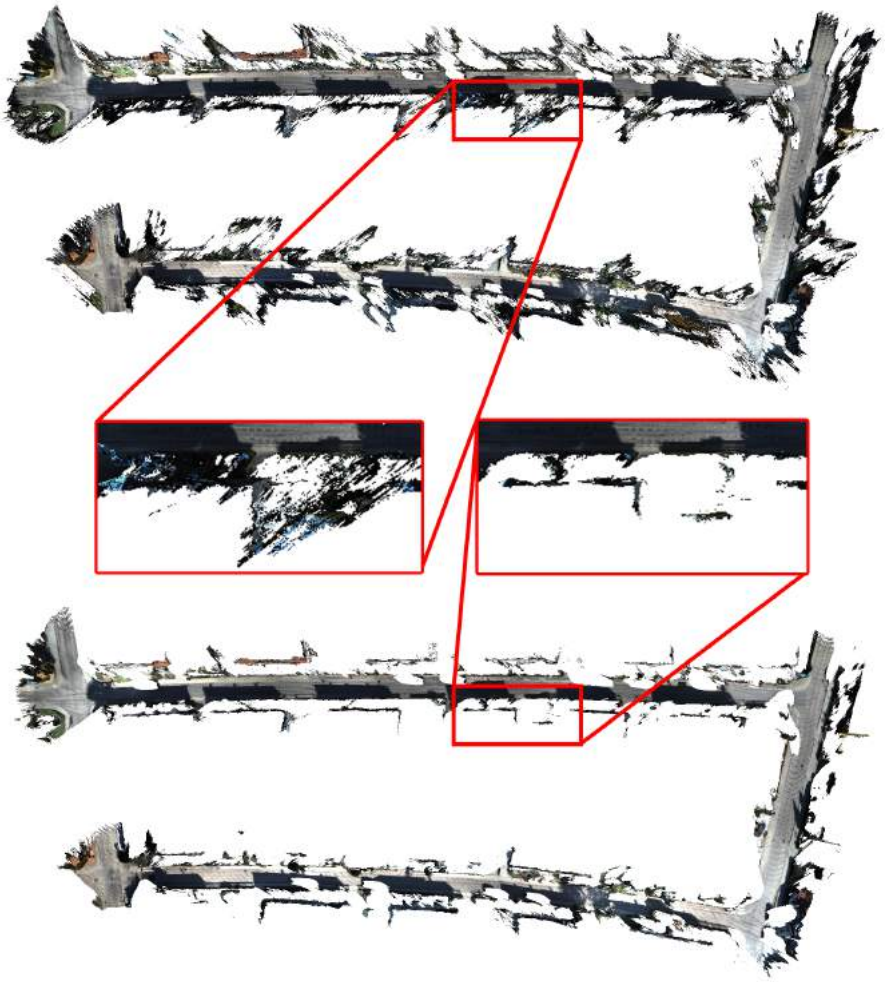


Figure 5.12: A comparison between a map of a residential area produced with regularization turned off (top), and one produced with regularization turned on (bottom, $k_{\text{weight}} = 4$). Note the much clearer outlines of the houses on the street in the regularized version. The middle section highlights an example where the regularization removes much of the streaking artifacts, while preserving the outline of the house and the surfaces of the road and sidewalk. Additionally, storing the regularized version uses less than half the amount of memory required to store the unregularized one.

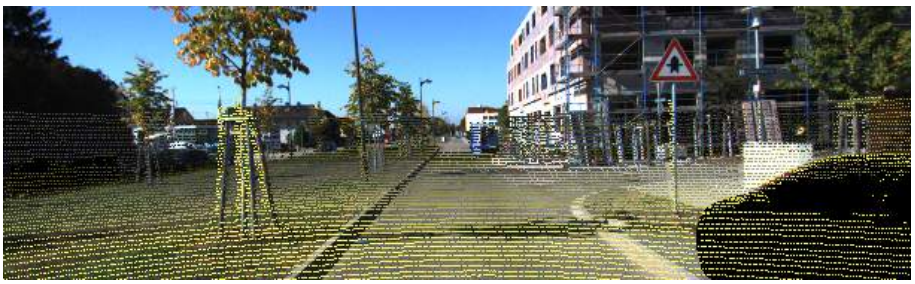


Figure 5.13: A sample KITTI frame with the available ground truth LIDAR points superimposed in yellow. Note the limited vertical range of the LIDAR, which doesn't reach the leaves of the tree on the left or the top of the sign on the right.

improvement. This is because the "streaks" in the 3D map tend to be oriented away from the moving camera, and, therefore, away from the LIDAR, preventing them from being compared to any ground truth in a meaningful way. Moreover, as can be seen in Figure 4.2, these artifacts are typically associated with objects such as trees, signs, and buildings, and are very often present above the upper range of the LIDAR (Figure 5.13), thus making them impossible to compare to any ground truth.

At the same time, it is also not possible to compensate for this limitation by relying on the backwards- and side-facing LIDAR readings from future frames. This is because they would significantly distort the evaluation metrics, since the reconstructions do not incorporate information coming from those angles. For example, the top of a tree on the side of the road is not shaped like a full sphere in the reconstruction, but like a half-sphere, since is only viewed from one side by the vehicle's front-facing cameras, as the they are moving towards it. Using backwards- and side-facing LIDAR data to evaluate the map accuracy is therefore not possible without distorting the metrics.

Nevertheless, we are able to show quantitative results indicating that map accuracy does exhibit a modest increase when regularization is performed, which we then complement with numerous additional qualitative results. Furthermore, we show that memory usage drops significantly with increased regularization strength, all while the map completeness is only affected to an acceptable extent.

Figure 5.14 shows the evolution of the system's memory usage over time, for different values of k_{weight}, the noise voxel weight threshold (see §4.5 for details).

As expected, the memory consumption of the system goes down as the noise threshold is set higher, that is, as voxels are more aggressively pruned from the map. The differentiation starts at frame 80, as that is the minimum pruning age we have set in our experiments (corresponding to eight seconds of driving time, enough to avoid pruning voxels still in view). At the same time, the lower density of the ELAS depth maps also leads to overall less memory usage. This aspect will be covered in more detail in the following paragraphs.

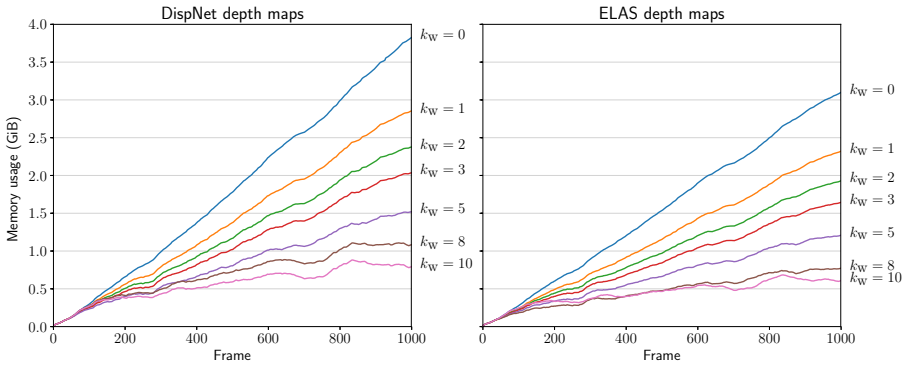


Figure 5.14: Memory consumption over time, under different voxel garbage collection thresholds ($k_w = k_{weight}$). A value of zero indicates no garbage collection. Higher values correspond to more aggressive garbage collection. In all examples, the minimum collection age was set to 80 frames.

Figure 5.15 illustrates the accuracy and completeness of the reconstructions as a function of k_{weight}, as well as the memory usage and an F1 score combining accuracy and completion.

The accuracy and completeness metrics are defined at the beginning of this chapter, in §5.1. For the purpose of evaluating the regularization, we also combine these two metrics into an F1-score, in order to better compare their interplay to the memory usage, as a function of the regularization strength. The F1-score is defined as

$$F_1(\text{frame}) = 2 \cdot \frac{A(\text{frame}) \cdot C(\text{frame})}{A(\text{frame}) + C(\text{frame})}, \tag{5.1}$$

where A and C represent the aforementioned accuracy and completeness metrics. The primary purpose of this metric is to define a meaningful way of combining reconstruction accuracy and completeness, in order to ease the task of evaluating the trade-offs between reducing memory consumption and reducing accuracy and completeness.

As illustrated in §3.2, the maps produced by DispNet are denser than those produced by ELAS, which is also reflected in the completeness of the reconstruction. At the same time, as seen in the previous sections, despite scoring less than DispNet on the KITTI Stereo Benchmark [49], depth maps produced by ELAS lead to more accurate, albeit less complete, reconstructions. This trend is maintained even with increasing k_{weight}.

Based on the F1-score plot from Figure 5.15, despite the fact that the reconstructions produced using ELAS are more accurate, and those using DispNet are more complete, their corresponding F1-scores are similar for values of k_{weight} smaller than 5. Afterwards, the DispNet reconstruction beings gaining a noticeable advantage over the ELAS one. A possible explanation of this trend is the fact that DispNet can draw more benefits from aggressive pruning than ELAS. As k_{weight} increases, the regularization reduces the impact of DispNet's downsides, namely, the stronger streaking artifacts and softness,

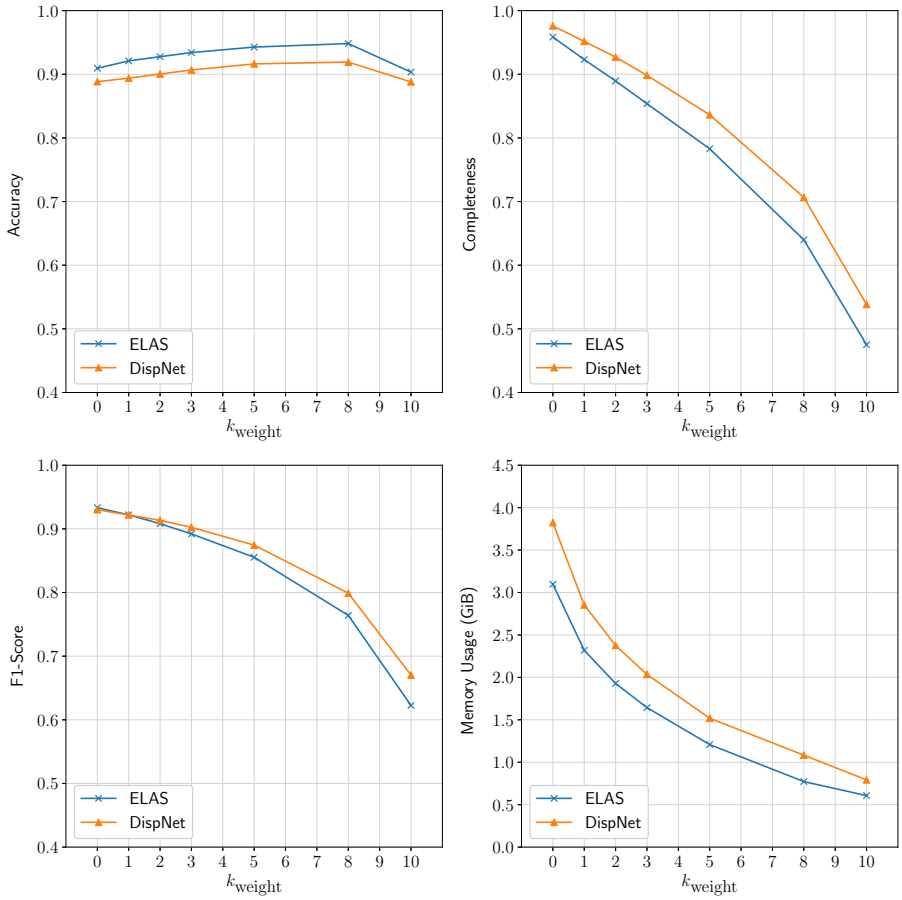


Figure 5.15: Reconstruction accuracy, completeness, F1-score, and memory usage (in GiB) as functions of the regularization strength, comparing its effect on reconstructions using DispNet and ELAS depth maps. Larger values of k_{weight} correspond to stronger regularization (i.e., more aggressive voxel garbage collection).

without seriously impacting its strong points, such as its density and its robustness to reflective and transparent surfaces.

Finally, based on the memory usage plot from Figures 5.14 and 5.15, it is clear that the impact of the noise on the memory consumption of the reconstructions is very pronounced. Even using very light regularizations with $k_{\text{weight}} = 1$ or 2 can already reduce the memory footprint of a reconstruction by more than 30%, with only small costs in terms of discarded (useful) information.

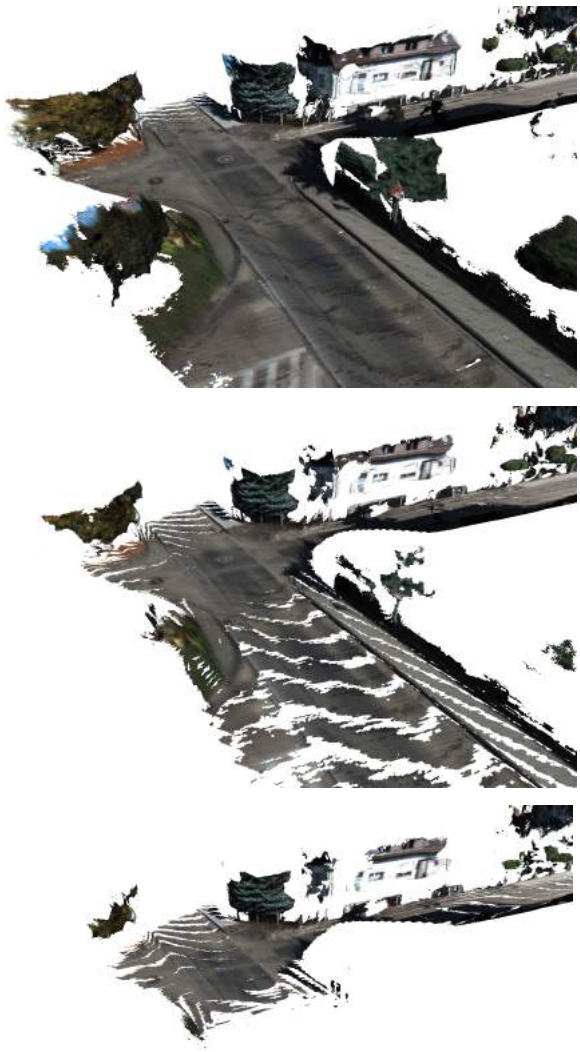
Figures 5.16 and 5.17 showcase the results of different aggressiveness levels of the voxel garbage collection on the static map. Figures 5.18, 5.19, and 5.20 show the results of the garbage collection on reconstructed cars.

(a) No voxel GC. Note the artifacts towards the left half of the reconstruction, consisting primarily of thick streaks extending away from the trees.

(b) $w_{\text{min}} = 3$. While noise is still present in the reconstruction, its impact is significantly reduced.

(c) $w_{\text{min}} = 5$. Most of the undesirable noise has been eliminated, while the important regions of the map (the road, fences, markings, etc. are still clearly visible).

Figure 5.16: Examples of reconstructions produced while using no voxel garbage and light/moderate GC, respectively. Both reconstructions use DispNet depth maps truncated at a maximum depth of 25m.



(a) $w_{\text{min}} = 8$. Nearly all the noise has been eliminated, but now essential parts of the reconstruction are starting to be affected. (Note the small gaps in the road surface.)

(b) $w_{\text{min}} = 15$. The gaps are getting larger, while nearly no noise is left to remove.

(c) $w_{\text{min}} = 20$. Most voxels end up being seen as noise and are collected, lending the map unusable.

Figure 5.17: Examples of very aggressive voxel garbage collection. Note that going past $w_{\text{min}} = 15$ is already much too aggressive for any practical purposes. Both reconstructions used DispNet depth maps truncated at a maximum depth of 25m.

Figure 5.18: Reconstructed car before and after voxel garbage collection (DispNet depth maps). This particularly challenging case was subject to strong motion, and erroneous measurement fusion for the first two frames of the track, leading to the heavy noise to the left of the car. The garbage collection is able to remove nearly all associated artifacts.

Figure 5.19: Reconstructed car before and after voxel garbage collection (DispNet depth maps). Note the clearer reconstruction border in the right image.

Figure 5.20: Reconstructed car before and after voxel garbage collection (ELAS depth maps). While the garbage collection does remove some of the undesired reconstruction artifacts, this image nevertheless highlights one of the general weaknesses of ELAS, namely its inability to work well with transparent surfaces, leading to the back window of the car appearing like a horizontal surface.

5.3 Tracking Accuracy

We perform a series of experiments to measure the impact of the direct alignment-based refinement stage of the 3D object tracking. Our results show that this process does not improve tracking performance, and, in fact, can even lead to less accurate results than the coarse stage alone.

The experiments are based on the KITTI tracking benchmark methodology [19]. However, given that DynSLAM does not estimate 3D bounding boxes for the dynamic objects, we only evaluated the accuracy of the relative pose estimation.

The ground truth information from the KITTI tracking benchmark consists of annotated 3D boxes, called *tracklets*, which are provided for all potentially dynamic objects in a sequence. For the scope of this evaluation, we focused on cars. The box coordinates are expressed in the camera's reference frame, and grouped into tracks. Therefore, computing the ground truth relative poses between consecutive frames is straightforward for all objects.

The analysis is performed as follows: for every car tracked by DynSLAM we find the matching ground truth tracklet, and compare the relative transformations between frames computed by DynSLAM to the ground truth relative transformations computed from the tracklet data. The error metrics are the same as those used in the KITTI odometry benchmark [19], that is, the length of the translation error, and the angle of the rotation error. The errors are computed on a frame-to-frame basis.

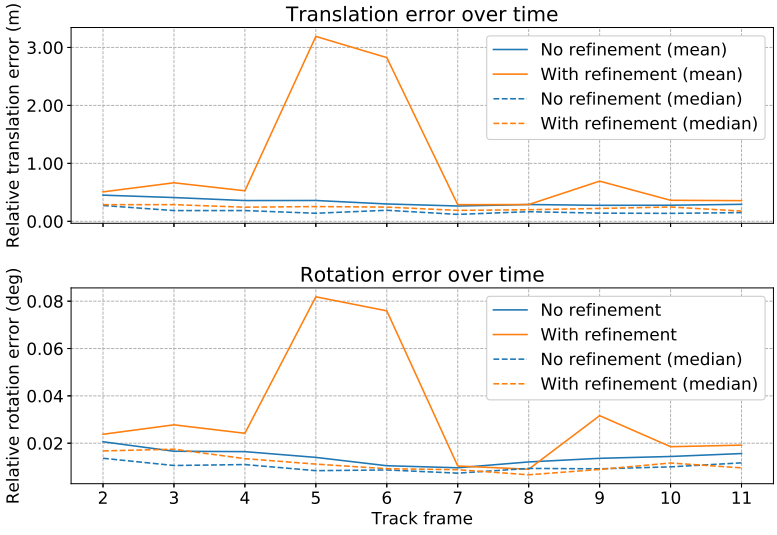


Figure 5.21: Aggregate 3D tracking errors with and without using the direct alignment refinement described in §4.3.2.

Figure 5.21 compares the translation and rotation errors of the 3D tracking

with and without the refinement stage. The results are computed over all objects which DynSLAM succeeds in tracking in sequences 0–6 from the KITTI tracking dataset, totaling over 1800 frames of moderate and heavy traffic. We consider the first 10 relative poses of all the tracks, and compute the mean and median errors for every track frame. Note that the evaluation starts with the second frame since we are evaluating relative poses, which are not defined for the first frame in a track. In total, we aggregated data from 33 successfully tracked objects.

As mentioned at the beginning of this section, the direct alignment phase does not improve the vehicle tracking process and, in fact, leads to worse results in most cases. This is particularly clear between frames 4 and 7, where the refinement process increases both rotation and translation errors by almost an order of magnitude, on average.

Moreover, as can be seen when comparing the mean and median results, the refinement method is susceptible to converge to very bad local optima, leading to extreme outliers in terms of both translation and rotation error. This, in turn, increases the variance of the error metrics by a significant margin.

Figure 5.22: Sample car which the direct alignment method fails to track. Note the pronounced reflections on the windows and hood, as well as the shadow patterns on its side. Moreover, despite being quite close to the camera, the featured vehicle occupies an area of roughly 350×200 pixels, less than 10% of the entire frame.

The failure of this approach can be explained by the nature of the objects being tracked. Cars have highly non-lambertian surfaces, consisting almost entirely of reflective and transparent materials. This violates some of the assumptions made by the direct alignment method, which expects photometric consistency between frames. Further challenges include extreme scale differences between consecutive frames, in particular when attempting to reconstruct cars moving towards the camera, as well as the low amount of data, given that in many cases the tracked cars only occupy a fraction of the screen. Figure 5.22 shows an example of a challenging scenario in which the direct alignment method fails.

In the future we plan to make further improvements to the coarse method, before improving the refinement stage, since we consider the former to hold more untapped potential and "low-hanging fruit" than the latter. These ideas

are presented in more detail in §6.2.3.

5.4 Ablation Studies

5.4.1 Reduced Spatial Resolution

We analyze the impact of the input spatial resolution on the accuracy and run time of DynSLAM. Most of these experiments are performed on KITTI odometry sequence number 6, which consists of 1101 frames exhibiting a good balance of buildings, vegetation, and traffic.

Our findings reveal that, as would be expected, the reconstruction accuracy decreases with the input resolution. Similarly, the run time of some components, such as ELAS, also decreases when operating on lower-resolution input. At the same time, we found that even when using 25% of the input dimensions, i.e., 306×92 input, the system can still produce reasonable reconstructions, albeit only when using ELAS depth maps. The performance of DispNet drops significantly when using low-resolution input, leading to noisier reconstructions.

Table 5.3: Mean inference time for ELAS and MNC, the most time-consuming elements of our pipeline, as a function of the input resolution.

Resolution	**ELAS**		**MNC**	
100%	121ms	(std=7ms)	231ms	(std=5ms)
75%	71ms	(std=4ms)	229ms	(std=4ms)
50%	33ms	(std=3ms)	236ms	(std=8ms)
25%	8ms	(std=4ms)	235ms	(std=10ms)

Table 5.3 shows the inference times of ELAS and the Multi-task Network Cascades, which are by far the most expensive operations in our system, as a function of the input resolution. Note that while the computation time of ELAS does decrease with the resolution of its inputs, the time taken by the instance-aware semantic segmentation does not.

We conjecture that the reason behind this is the fixed number of object proposals (300, as mentioned in the original paper [13]) generated by the first stage and refined by the second one. Despite lowering the input resolution, and with it, the cost of computing the convolutional image features, the run time of the pipeline ends up being bounded by the proposal generation, ranking, and refinement, which are almost completely independent of the input size in terms of their computational costs.

Note that because of their generic nature, the processing times of the depth and instance-aware semantic segmentation components do not vary in significant ways across different KITTI sequences.

Figure 5.23 and Table 5.4 show the reconstruction accuracy of DynSLAM as a function of the input resolution. The metric used is the same as in §5.1,

Table 5.4: Reconstruction quality as a function of input resolution. The quality is computed as the mean of the disparity errors in the sequence's frames. See §5.1 for more details on evaluation metrics.

		ELAS		**DispNet**	
		Input	Fused	Input	Fused
100%	(1226×370)	0.0729	0.0724	0.0761	0.0807
75%	(919×277)	0.0643	0.0689	0.0836	0.1031
50%	(613×185)	0.0690	0.0807	0.1238	0.1602
25%	(306×92)	0.1162	0.1624	0.5206	0.5989

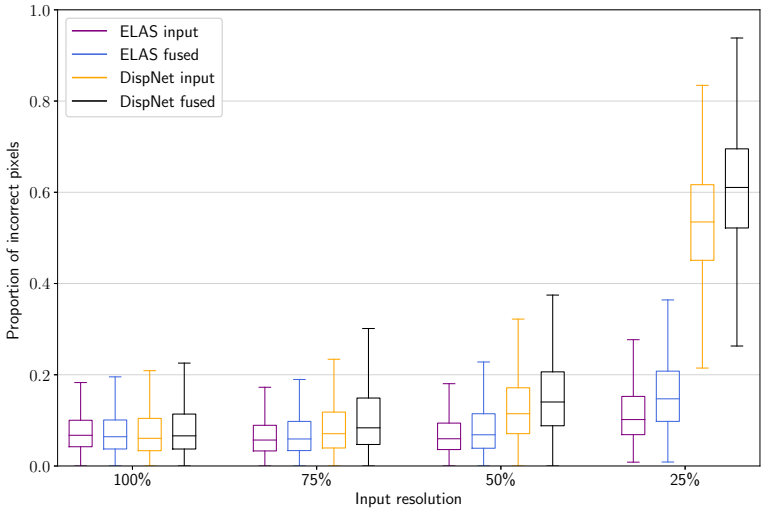


Figure 5.23: Input and reconstruction errors as functions of the input resolution. The evaluation was performed on sequence 6 from the KITTI odometry benchmark. The box plots follow the Tukey convention, as described earlier in this chapter.

namely, counting pixels whose disparity error is >3px and >5% of the ground truth disparity as inaccurate.

As expected, the general trend is for the reconstruction error to increase as the resolution of the input is decreased. Nevertheless, for ELAS-based reconstruction, we notice a small increase in accuracy at 75%, as compared to 100% resolution. A possible explanation for this result is the fact that reducing the depth map resolution acts as a soft regularizer, reducing the impact of small bumps and other artifacts on the overall reconstruction quality. 75% resolution could therefore be seen as a "sweet spot" for good reconstructions, by reducing high-frequency noise associated with high-resolution depth maps, while at the same time having sufficient resolution to produce a faithful reconstruction.

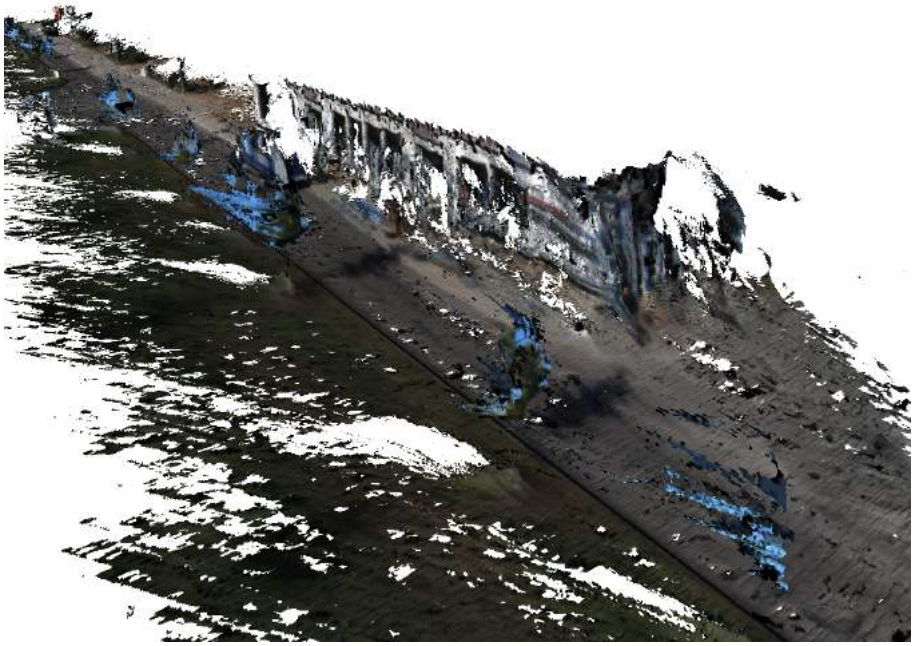
Another interesting effect is the fact that as resolution decreases, the reconstruction (fused) error increases faster than the input error. This is due to the accumulation of errors in the map: if the depth maps become too degraded, artifacts begin to accumulate in the map, leading to errors over multiple subsequent frames. That is, an isolated erroneous (but large) bump in a depth map is registered as an input error only in the frame in which it is present. However, once it gets fused into the map, it might take several frames until subsequent measurements "smooth it out".

In other words, the regularizing effect of fusing depth maps across multiple frames can start to backfire when the quality of individual depth maps drops below a certain threshold, leading to severe map corruption caused by the cascading effects of artifacts from different input frames accumulating in the map. As is clearly visible in Figure 5.23, this effect is much more pronounced when using DispNet depth maps. This can be interpreted as a reflection of the fact that the DispNet architecture was only trained using high-resolution data. ELAS, not being reliant on training data, does not exhibit this problem, leading to reasonable results even at 306×92 resolution, i.e., less than QVGA.

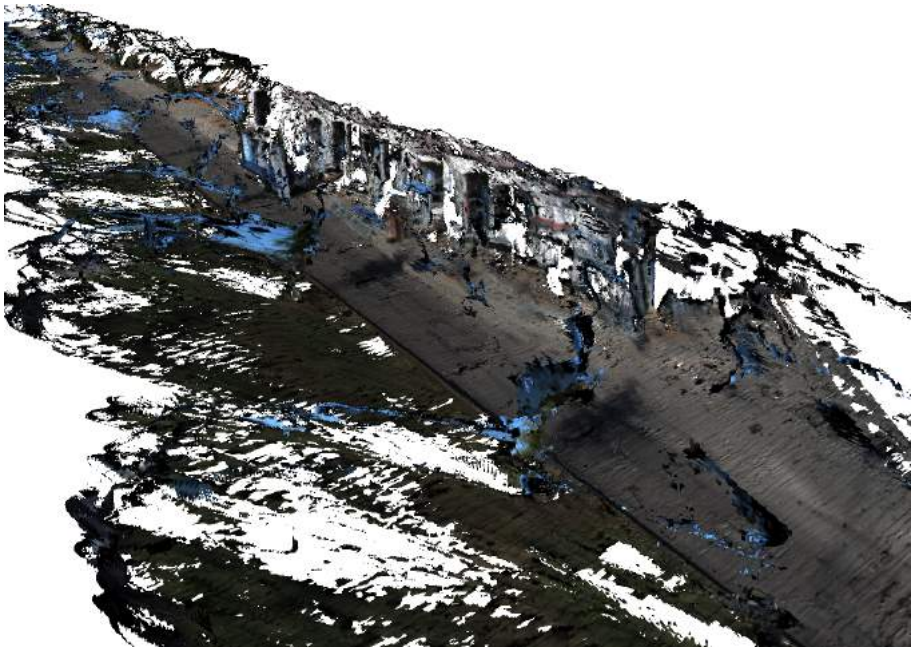
Figure 5.24 shows a comparison between a reconstruction produced by our system from reduced resolution (25%) input using ELAS depth maps, and using DispNet depth maps. Note that while somewhat more sparse, the reconstruction produced using ELAS is also less noisy and suffers from considerably less distortion than the DispNet one. This is in accordance with the quantitative results presented in Figure 5.23.

Figures 5.25 and 5.26 show comparisons between reconstructions computed from full- and low-resolution input, using DispNet and ELAS depth maps, respectively. ELAS depth maps produce overall sharper maps, even at low resolution, while DispNet leads to distortion in numerous places, such as the walls of the house. Note the circled car from the low-resolution reconstructions. Its presence reflects the limitations of the instance-aware semantic segmentation, whose false negative rate increases substantially when working with low-resolution input.

Similarly, Figure 5.27 shows vehicles reconstructed by DynSLAM under the same configurations. Note that despite being incomplete and moderately distorted, the ELAS-based reconstruction is clearer than the DispNet one, which is nearly unrecognizable as a car due to the extensive noise. Note that



(a) Reconstructed map using ELAS depth maps at 25% resolution.



(b) Reconstructed map using DispNet depth maps at 25% resolution.

Figure 5.24: A comparison of maps computed using reduced-resolution input. Note that while both reconstructions are of significantly lower quality than those produced from full-resolution input, the ELAS map is considerably less noisy than the DispNet one, which exhibits strong distortions, especially on the building's facade.

(a) DispNet depth maps, full resolution

(b) DispNet depth maps, 0.75× resolution.

(c) DispNet depth maps, 0.50× resolution.

(d) DispNet depth maps, 0.25× resolution.

Figure 5.25: DispNet-based reconstructions at full- and low-resolution. The circled car represents a false negative of the instance-aware semantic segmentation component, caused by lower-resolution input.

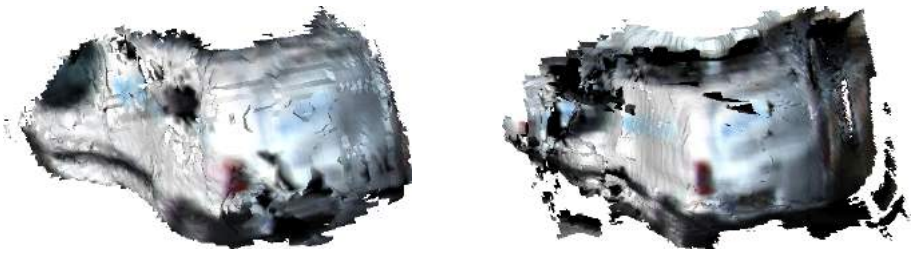
(a) ELAS depth maps, full resolution

(b) ELAS depth maps, 0.75× resolution.

(c) ELAS depth maps, 0.50× resolution.

(d) ELAS depth maps, 0.25× resolution.

Figure 5.26: ELAS-based reconstructions at full- and low-resolution. The circled car represents a false negative of the instance-aware semantic segmentation component, caused by lower-resolution input.



(a) Vehicle reconstructed using ELAS depth maps at 25% resolution.

(b) Vehicle reconstructed using DispNet depth maps at 25% resolution.

Figure 5.27: A comparison of vehicle reconstructions attempted using reduced-resolution input.

these reconstructions, much like those from Figure 5.24, have been pruned by the voxel garbage collection. Nevertheless, the DispNet reconstruction was too distorted for the pruning process to have any meaningful effect.

In conclusion, we found that using ELAS to compute depth maps allows DynSLAM to function with acceptable accuracy even on very low-resolution input. On the other hand, while capable of producing qualitatively superior vehicle reconstructions on full-resolution input, DispNet depth maps lead to very poor performance on low-resolution (e.g., 25%) input.

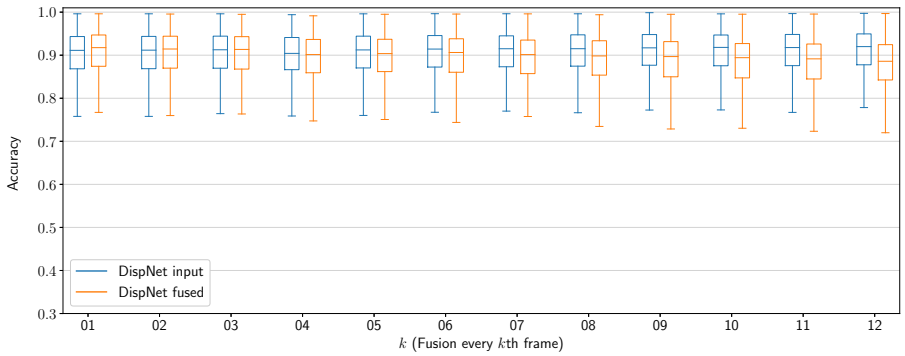
5.4.2 Reduced Temporal Resolution

Given that the run time of our pipeline is dominated by the instance-aware semantic segmentation phase which, as seen in the previous subsection, takes roughly 250ms irrespective of the spatial resolution of the input, we present a series of preliminary experiments investigating the possibility of avoiding to run the computationally expensive segmentation every frame.

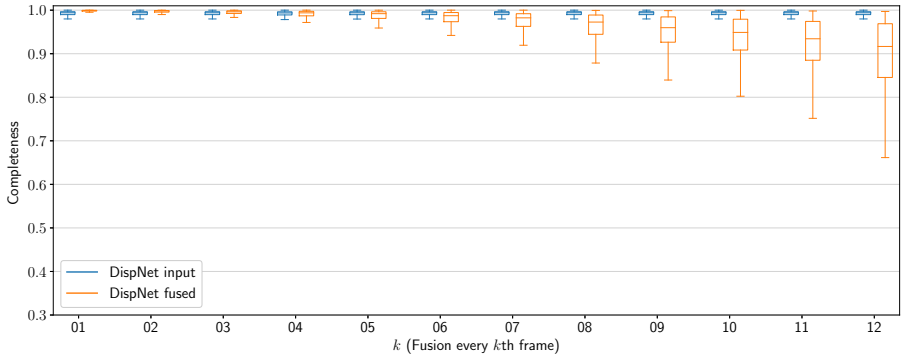
In other words, we consider the option of running cheaper components such as visual odometry on every input frame, but only computing dense depth maps, the instance-aware semantic segmentation, and the static map fusion every k frames, with the hope that it would allow the system to operate closer to real time, without significant losses in terms of mapping accuracy and completeness.

Early experiments showed that the 3D object tracking performs rather poorly even when it is run every two frames instead of every frame, but that the static map fusion remains accurate even when performed every 3–5 frames. Because of this behavior, we choose to focus on evaluating the quality of the dense depth map, using the non-semantic (i.e., full-frame) method described in §5.1. As such, we use the first 1000 frames of KITTI odometry sequence 09, which contain almost no dynamic objects.

The results of this experiment are shown in Figure 5.28. As expected, on its own, the accuracy of the reconstruction is not affected since the input depth maps are themselves accurate. On the other hand, the reconstruction



(a) Reconstruction accuracy as a function of k, the frequency at which fusion is performed.



(b) Reconstruction completeness as a function of k, the frequency at which fusion is performed.

Figure 5.28: The impact of reduced temporal resolution on static map reconstruction accuracy and completeness. A value of $k = 1$ signifies fusion performed every frame (the default case), $k = 2$, every two frames, etc.

completeness score starts to drop significantly once fusion is performed more rarely than every 4–5 frames.

In conclusion, the high completeness and accuracy scores obtained even when performing the static map fusion every two or three frames show that it is not necessary to perform this operation at input frequency. In the future, this insight could be used to improve the efficiency of similar systems, prioritizing tasks such as object tracking and planning to run on every frame, but performing auxiliary tasks such as mapping with a lower frequency.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The high-level aim of this thesis was to explore ways of building richer and more robust maps of complex environments, with the end goal of empowering further applications, such as three-dimensional object tracking, motion planning in urban environments, and detailed real-time visualizations, thereby bringing fully autonomous vehicles one step closer to being a reality.

To this end, we designed and implemented DynSLAM, a novel dense mapping pipeline based on InfiniTAM [38] which is capable of robust operation in challenging urban environments. In addition to building a static map of its environment, DynSLAM also detects, tracks, and densely reconstructs the dynamic objects encountered within.

Our pipeline achieves this by computing instance-aware semantic segmentations of its input frames, and using them to separate potentially dynamic objects, such as vehicles, from the background. It then uses the sparse scene flow to analyze the motion of the objects relative to the camera, classifying them as static or dynamic, and reconstructing them in the process.

We also discussed the artifacts caused by the noise associated with estimating depth from stereo and presented a technique for reducing their impact, thereby increasing the quality of the reconstructions produced by our system, while also reducing their memory consumption by a significant margin.

We presented thorough quantitative and qualitative evaluations of our system on a large number of real-world video sequences from the KITTI odometry and tracking [19] benchmarks, comparing the performance and run time of different configurations, and highlighting the system's robustness to adverse conditions such as heavy traffic, sudden lighting changes, and fast motion.

To the best of our knowledge, the system presented in this thesis is the first dense mapping pipeline capable of simultaneously reconstructing the objects encountered in an outdoor environment, in addition to the static map, in near-real-time, while using only stereo input.

6.2 Future Work

While developing DynSLAM, many possible avenues of improvement had to be left out, in order to ensure that the core contributions were implemented and evaluated in a thorough manner. This section will provide an overview of some of the many ways in which DynSLAM's performance, run-time, and robustness could be improved in the future.

6.2.1 Performance Improvements

As discussed in Chapter 5, the main bottleneck of our system is the instance-aware semantic segmentation, which is incapable of running at more than 4–5Hz, even on high-end hardware. Given that without this component, our system can easily run at over 10Hz, any work towards improving end-to-end performance should start here. Neural network architectures for non-instance-aware semantic segmentation, such as ENet [58] can already run at over 40Hz. Extending such architectures to perform instance-aware segmentation could result in improved performance for DynSLAM.

Another possible trade-off worth investigating is using batching for CPU-heavy operations, such as estimating disparity from stereo using ELAS. Given that the implementation is single-core, it would be possible to increase the system's throughput by computing 2–3 depth maps in parallel, while also introducing 1–2 frames of (acceptable) latency to the system.

6.2.2 Uncertainty Propagation

DynSLAM currently uses a simplistic model for measurement uncertainty, which is formulated as a linear function of the inverse depth. Despite being reasonably robust for our current use case, a more elaborate model could help produce even more accurate reconstructions, reducing, or possibly eliminating, the need for map regularization.

While neither ELAS nor DispNet are capable of associating confidence scores to the values in the computed disparity maps, methods such as [45] estimate probability distributions over possible disparity values for every pixel, which could potentially be passed on to future pipeline stages in order to more rigorously model the voxel-wise confidence in the resulting reconstruction.

6.2.3 3D Pose Estimation

There are also numerous ways of improving the accuracy of the 2D and 3D object tracking, which can lead to improved reconstructions. Currently, all pose estimation procedures occur on a frame-to-frame basis. Augmenting DynSLAM to track features across multiple frames, and to optimize object trajectories over longer windows has the potential to yield improved reconstructions. Moreover, multi-frame methods could also enable support for tracking objects across longer occlusions than is currently possible in DynSLAM.

Neural network-based methods for direct rigid body motion estimation from point clouds, such as the **SE3-Nets** proposed by Byravan and Fox [7], would be another approach worth exploring.

6.2.4 Mapping

DynSLAM does not support loop closure detection and global map optimization. While challenging to perform in the context of volumetric representations, map adjustments based on global optimization are still a key aspect of building quality large-scale maps, and an important direction for further development of DynSLAM. Kähler et al. [34] have shown how to extend the InfiniTAM framework to enable loop closure detection and global refinement using a submap-based approach.

Semi-dense SLAM methods [15] have shown great promise in terms of accuracy, scalability, and speed. Despite the fact that semi-dense maps are less information-rich than dense ones, hybrid approaches which use, e.g., dense representations for the roads and cars, but semi-dense ones for the rest of the environment (trees, houses, and distant objects) may be able to effectively bring out the best of both worlds.

6.2.5 Evaluation

As discussed in Chapter 5, the methods used for evaluating our system's performance could be improved in a number of ways.

First, due to its inherent nature, the KITTI dataset does not provide any ground truth information on the shape of the dynamic objects apart from the LIDAR, making the evaluation of their reconstructions difficult. This issue could be addressed in the future by using either simulated datasets, such as Virtual KITTI [18], or custom real-world datasets with ground truth object shape information produced via precise laser scanning, following the work of Rünz and Agapito [46].

Second, as described in [49], evaluating disparity maps based on LIDAR ground truth in the camera's frame is not optimal due to occlusions. That is, certain objects directly visible to the LIDAR may not be visible to the camera, resulting in systematic evaluation bias.

Possible ways of addressing this limitation include either manually adjusting the ground truth LIDAR data to account for occlusions, or performing the evaluation inside the LIDAR's reference frame, as opposed to the camera's. The former method is used as part of the standard KITTI Stereo benchmark [49], and has the potential to be highly reliable[1]. Nevertheless, this approach is time-consuming, making it infeasible for correcting long sequences consisting of thousands of frames, which are required for evaluating large-scale mapping systems such as DynSLAM in a realistic manner. The second method would involve computing virtual LIDAR-like point clouds of the reconstruction by adapting the raycasting component of InfiniTAM to function similar to a LIDAR. This would therefore allow both the input depth frames, as well as the fused map to be rendered from the point of view of the LIDAR, allowing them to be compared directly to the ground truth, eliminating the issues caused by occlusion.

[1] Note that the KITTI Stereo 2015 benchmark only contains 200 training and 200 test images.

6.2.6 Segmentation

As discussed in Chapter 5, relying purely on semantic information for object detection is not always reliable. The Multi-task Network Cascades [13] model we employ often fails to detect bikes, vans, and larger trucks, especially in difficult lighting conditions and under reduced resolution.

From a mapping perspective, if the undetected objects are not moving, they are simply fused into the static map, without causing serious problems. However, if moving, they can corrupt the map, leaving behind noisy trails, as if the reconstruction were performed using the standard fusion mode described in §5.1.3, defeating the purpose of DynSLAM.

For this reason, the development of methods for effectively combining both semantic, as well as motion cues for object detection is a promising area of future research.

Bibliography

[1] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. Measuring the objectness of image windows. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2189–2202, 2012. [Cited on page 28.]

[2] Rares Ambruş, Nils Bore, John Folkesson, and Patric Jensfelt. Meta-rooms: Building and maintaining long term spatial models in a dynamic world. *IEEE International Conference on Intelligent Robots and Systems*, (Iros):1854–1861, 2014. [Cited on pages 2 and 15.]

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015. [Cited on pages 11 and 13.]

[4] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (SLAM): Part I. *IEEE Robotics and Automation Magazine*, 13(3):108–117, 2006. [Cited on page 24.]

[5] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Computer vision and image understanding*, 110(3):346–359, 2008. [Cited on page 18.]

[6] Charles Bibby and Ian Reid. A hybrid SLAM representation for dynamic marine environments. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 257–264, 2010. [Cited on page 2.]

[7] Arunkumar Byravan and Dieter Fox. SE3-Nets: Learning Rigid Body Motion using Deep Neural Networks. page 8, 2016. [Cited on pages 11 and 80.]

[8] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Ian D Reid, and John J Leonard. Past , Present , and Future of Simultaneous Localization And Mapping : Towards the Robust-Perception Age. 32(6):1–27, 2016. [Cited on page 24.]

[9] Florian Chabot, Mohamed Chaouch, Jaonary Rabarisoa, Céline Teulière, and Thierry Chateau. Deep MANTA: A Coarse-to-fine Many-Task Network for joint 2D and 3D vehicle analysis from monocular image. 2017. [Cited on page 11.]

[10] Wongun Choi. Near-online multi-target tracking with aggregated local flow descriptor. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3029–3037, 2015. [Cited on page 11.]

[11] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996. [Cited on page 25.]

[12] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration. *Popl*, pages 1–16, apr 2016. [Cited on page 9.]

[13] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware Semantic Segmentation via Multi-task Network Cascades. *arXiv:1512.04412*, pages 3150–3158, 2015. [Cited on pages 3, 11, 12, 13, 29, 30, 35, 71, and 82.]

[14] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *IEEE International Conference on Computer Vision*, pages 2758–2766, 2015. [Cited on page 23.]

[15] Jakob Engel, Thomas Schöps, and Daniel Cremers. *LSD-SLAM: Large-Scale Direct Monocular SLAM*, pages 834–849. Springer International Publishing, Cham, 2014. [Cited on page 81.]

[16] Thomas Faeulhammer, Rares Ambrus, Christopher Burbridge, Michael Zillich, John Folkesson, Nick Hawes, Patric Jensfelt, Markus Vincze, Thomas Fäulhammer, Christopher Burbridge, Michael Zillich, John Folkesson, Nick Hawes, Patric Jensfelt, and Markus Vincze. Autonomous Learning of Object Models on a Mobile Robot. *IEEE Robotics and Automation Letters*, 2(1):1, 2016. [Cited on pages 2, 14, and 15.]

[17] James Fung and Steve Mann. Computer vision signal processing on graphics processing units. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 5, pages V–93. IEEE, 2004. [Cited on page 30.]

[18] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016. [Cited on pages 16 and 81.]

[19] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012. [Cited on pages 3, 35, 39, 49, 69, and 79.]

[20] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Asian conference on computer vision*, pages 25–38. Springer, 2010. [Cited on pages 3, 6, 21, and 35.]

[21] Andreas Geiger, Julius Ziegler, and Christoph Stiller. StereoScan: Dense 3d reconstruction in real-time. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 963–968. IEEE, jun 2011. [Cited on pages 5, 6, 17, 35, 38, and 43.]

[22] Bernardes Vitor Giovani, Alessandro Correa Victorino, and Janito Vaqueiro Ferreira. Stereo Vision for Dynamic Urban Environment Perception Using Semantic Context in Evidential Grid. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2015-Octob:2471–2476, 2015. [Cited on page 36.]

[23] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. [Cited on page 12.]

[24] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. [Cited on page 12.]

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. [Cited on page 23.]

[26] Fatma Guney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4165–4175, 2015. [Cited on page 6.]

[27] Bharath Hariharan, Pablo Arbeláez, Ross Girshick, and Jitendra Malik. Simultaneous detection and segmentation. In *European Conference on Computer Vision*, pages 297–312. Springer, 2014. [Cited on page 12.]

[28] David Held, Jesse Levinson, Sebastian Thrun, and Silvio Savarese. Robust real-time tracking combining 3D shape, color, and motion. *The International Journal of Robotics Research*, 35(1-3):1–28, 2015. [Cited on page 10.]

[29] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2010. [Cited on page 7.]

[30] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2008. [Cited on page 6.]

[31] A Huang, N Roy, A Bachrach, P Henry, M Krainin, D Maturana, and D Fox. Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. *Springer Tracts in Advanced Robotics*, 100:235–252, 2011. [Cited on page 5.]

[32] ROS Industrial. 3d camera survey, http://rosindustrial.org/news/2016/1/13/3d-camera-survey, 2016. [Cited on page 43.]

[33] Cansen Jiang, Danda Pani Paudel, Yohan Fougerolle, David Fofi, and Cedric Demonceaux. Static-Map and Dynamic Object Reconstruction in Outdoor Scenes Using 3-D Motion Segmentation. *IEEE Robotics and Automation Letters*, 1(1):324–331, 2016. [Cited on pages 2, 3, 14, and 15.]

[34] Olaf Kähler, Victor A. Prisacariu, and David W. Murray. Real-Time Large-Scale Dense 3D Reconstruction with Loop Closure. Number 4 in Lecture Notes in Computer Science, pages 500–516. Springer International Publishing, Cham, nov 2016. [Cited on pages 9 and 81.]

[35] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion. *3Dv*, pages 1–8, 2013. [Cited on page 7.]

[36] Deyvid Kochanov, Aljosa Osep, Jorg Stuckler, and Bastian Leibe. Scene flow propagation for semantic mapping and object discovery in dynamic street scenes. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (3):1785–1792, 2016. [Cited on pages 2, 14, and 15.]

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. [Cited on pages 11 and 30.]

[38] Olaf Kähler, Victor Adrian Prisacariu, Carl Yuheng Ren, Xin Sun, Philip Torr, and David Murray. Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1241–1250, 2015. [Cited on pages 3, 8, 9, 25, 28, 36, 41, 58, and 79.]

[39] Karel Lebeda, Simon Hadfield, and Richard Bowden. 2D or not 2D: Bridging the gap between tracking and structure from motion. pages 642–658, 2014. [Cited on pages 3 and 10.]

[40] Karel Lebeda, Simon Hadfield, and Richard Bowden. Dense Rigid Reconstruction from Unstructured Discontinuous Video. In *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, volume 2016-Febru, pages 814–822. IEEE, dec 2015. [Cited on page 10.]

[41] Philip Lenz, Julius Ziegler, Andreas Geiger, and Martin Roser. Sparse scene flow segmentation for moving object detection in urban environments. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, volume 44, pages 926–932. IEEE, jun 2011. [Cited on pages 11 and 36.]

[42] Peidong Liu, Lionel Heng, Torsten Sattler, Andreas Geiger, and Marc Pollefeys. Direct Visual Odometry for a Fisheye-Stereo Camera. 2017. [Cited on pages 40 and 41.]

[43] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015. [Cited on page 11.]

[44] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. [Cited on pages 18 and 20.]

[45] Wenjie Luo, Alexander G. Schwing, and Raquel Urtasun. Efficient Deep Learning for Stereo Matching. *Conference on Computer Vision and Pattern Recognition*, pages 5695–5703, 2016. [Cited on page 80.]

[46] Martin Rünz and Lourdes Agapito. Co-Fusion : Real-time Segmentation, Tracking and Fusion of Multiple Objects. *Icra*, pages 4471–4478, 2017. [Cited on pages 14, 16, and 81.]

[47] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. pages 4040–4048, 2015. [Cited on pages 7, 21, 22, 30, and 35.]

[48] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks. *Bayesian Forecasting and Dynamic Models*, 22(2):1–694, sep 2016. [Cited on pages 7 and 8.]

[49] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. pages 3061–3070, 2015. [Cited on pages 6, 22, 52, 53, 64, and 81.]

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

[51] Raúl Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 2017. [Cited on page 5.]

[52] Richard A Newcombe, Dieter Fox, and Steven M Seitz. DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015. [Cited on page 8.]

[53] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. *2011 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2011*, pages 127–136, 2011. [Cited on pages 8, 25, and 30.]

[54] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. DTAM: Dense tracking and mapping in real-time. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2320–2327, 2011. [Cited on page 24.]

[55] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Trans. Graph.*, 32(6), 2013. [Cited on pages 3, 8, 9, 14, 25, 27, 43, 44, and 47.]

[56] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. Ieee, 2004. [Cited on page 17.]

[57] Peter Ondruska, Pushmeet Kohli, and Shahram Izadi. MobileFusion: Real-Time Volumetric Surface Reconstruction and Dense Tracking on Mobile Phones. *IEEE Transactions on Visualization and Computer Graphics*, 21(11):1251–1258, 2015. [Cited on page 8.]

[58] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. *Iclr2017*, pages 1–10, 2016. [Cited on pages 12 and 80.]

[59] Hanspeter Pfister, Matthias Zwicker, Jeroen Van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co., 2000. [Cited on page 7.]

[60] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313, 1989. [Cited on page 1.]

[61] Victor Adrian Prisacariu, Olaf Kähler, Stuart Golodetz, Michael Sapienza, Tommaso Cavallari, Philip HS Torr, and David W Murray. Infinitam v3: A framework for large-scale 3d reconstruction with loop closure. *arXiv preprint arXiv:1708.00783*, 2017. [Cited on page 9.]

[62] N Dinesh Reddy, Prateek Singhal, Visesh Chari, and K Madhava Krishna. Dynamic Body VSLAM with Semantic Constraints. pages 1897–1904, 2015. [Cited on page 14.]

[63] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. [Cited on pages 12 and 29.]

[64] Akihito Seki and Marc Pollefeys. SGM-Nets: Semi-global matching with neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (1):231–240, 2017. [Cited on page 7.]

[65] Sunando Sengupta, Eric Greveson, Ali Shahrokni, and Philip HS Torr. Urban 3d semantic modelling using stereo vision. pages 580–585, 2013. [Cited on pages 2 and 52.]

[66] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. [Cited on page 11.]

[67] Randall C Smith and Peter Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56–68, 1986. [Cited on page 23.]

[68] Jörg Stückler and Sven Behnke. Model learning and real-time tracking using multi-resolution surfel maps. In *AAAI*, 2012. [Cited on page 7.]

[69] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013. [Cited on page 12.]

[70] V. Vineet, O. Miksik, M. Lidegaard, M. Nießner, S. Golodetz, V. A. Prisacariu, O. Kähler, D. W. Murray, S. Izadi, P. Pérez, and P. H. S. Torr. Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 75–82, May 2015. [Cited on pages 14, 25, 43, and 52.]

[71] Christoph Vogel, Konrad Schindler, and Stefan Roth. Piecewise rigid scene flow. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1377–1384, 2013. [Cited on page 15.]

[72] Trung-Dung Vu, Olivier Aycard, and Nils Appenrodt. Online Localization and Mapping with Moving Object Tracking in Dynamic Outdoor Environments. *Intelligent Vehicles Symposium, 2007 IEEE*, pages 190–195, 2007. [Cited on page 2.]

[73] Thibaut Weise, Thomas Wismer, Bastian Leibe, and Luc Van Gool. In-hand scanning with online loop closure. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops 2009*, number April, pages 1630–1637, 2009. [Cited on page 7.]

[74] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J Leonard, and John McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *IEEE International Conference on Robotics and Automation, ICRA*, 2013. [Cited on page 8.]

[75] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, Thomas Whelan, John Mcdonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J Leonard. Kintinuous : Spatially Extended KinectFusion. 2012. [Cited on page 8.]

[76] Thomas. Whelan, Michael. Kaess, Hordur. Johannsson, Maurice. Fallon, John. J. Leonard, and John. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2014. [Cited on pages 8, 9, and 26.]

[77] Thomas Whelan, Stefan Leutenegger, Renato F Salas-moreno, Ben Glocker, and Andrew J Davison. ElasticFusion : Dense SLAM Without A Pose Graph. *Robotics: Science and Systems*, 2015(December), 2015. [Cited on pages 7, 8, and 9.]

[78] Artur Wilkowski, Tomasz Kornuta, Maciej Stefańczyk, and Włodzimierz Kasprzak. Efficient generation of 3D surfel maps using RGB-D sensors. *International Journal of Applied Mathematics and Computer Science*, 26(1):99–122, 2016. [Cited on page 8.]

[79] Jure Žbontar and Yann Le Cun. Computing the stereo matching cost with a convolutional neural network. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 07-12-June(1):1592–1599, 2015. [Cited on page 7.]

[80] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. Octree-based fusion for realtime 3d reconstruction. *Graphical Models*, 75(3):126–136, 2013. [Cited on pages 8 and 9.]

[81] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008. [Cited on page 37.]

[82] Lv Zhaoyang. Kinfuseg: a dynamic slam approach based on kinect fusion. *Department of Computing, Imperial College London, London, England, Msc Thesis*, 2013. [Cited on pages 3, 14, and 15.]

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

SIMULTANEOUS LOCALIZATION AND MAPPING IN DYNAMIC ENVIRONMENTS

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
BÂRSAN	IOAN ANDREI

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 27.08.2017	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.