DOE-SLAM: Dynamic Object Enhanced Visual SLAM

by

Xiao Hu

Thesis submitted to the
Faculty of Engineering
In partial fulfillment of the requirements
For the M.C.S degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

A notable limitation of feature based vision simultaneous localization (vSLAM) in dynamic environments is the disastrous drift of the position estimate resulting in a complete loss of localization. State-of-the-art dynamic monocular vSLAM methods mask out all foreground objects and only use background features. This improves accuracy but also reduces the number of usable features in many scenes leading to unstable tracking. Instead, we formulate a novel strategy for monocular vSLAM that uses moving objects in the scene to improve accuracy, and extend ORB-SLAM2 to adapt to dynamic environments, estimating not only the camera trajectory based on background features but also foreground object motion. In the case where there are not enough background features for tracking, our method can use the features from the object and the prediction of the object motion to approximate the camera pose. We evaluate our system on various datasets, and our analysis shows that we achieve better pose estimation accuracy and robustness over state-of-the-art monocular vSLAM systems.

Acknowledgements

I would like to thank my supervisor, Professor Jochen Lang for his kindness help. His observant guidance and his supports in not only study requirements but also academic suggestions through my whole master's research life are invaluable and help me finally achieve my goal and my thesis.

Besides, I would like to appreciate my colleagues, lab mates and also my friends in VIVA lab for their amiable comments on my research and thesis work. Also, I would like to express my gratitude to my parents for their support of my study and encouragement to let me persist in my research.

Declaration

In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of University of Ottawa's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to IEEE website to learn how to obtain a License from RightsLink.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation of the problem

We live in a 3D world full of dynamic objects. Computer vision techniques are being developed to help computers and robots to see the real world in an intelligent manner. Augmented reality (AR) allows the user to observe the real world composited or superimposed with virtual information [2]. Different from virtual reality (VR), AR supplements reality, instead of completely replacing it. Thus, AR can enhance the way a user is able to interact with the real world. Van Krevelen *et al.* [87] survey the history of AR development. The first AR prototype was created in the 1960s by Ivan Sutherland and his students. The term "Augmented reality" is proposed by Thomas and David [83] in the early 1990s. A few years later, computing ability had been sufficiently improved, and computing devices were small enough to support graphical overlay in mobile settings. From then on, AR became a distinct field of research. Nowadays, AR applications have been explored in many different fields and applications are deployed [2].

The basic idea of AR is inserting virtual information into the real world to provide the user with extra knowledge about the scene [47]. One of the main challenges of AR is how to align the virtual information with the real world within the device (the camera or other sensors) frame. Until the early 2000s, most of the vision-based registration methods relied on markers [47]. Later on, with the development of keypoint extraction techniques and the multi-view geometry theory, some markerless methods appeared. The device localization and the environment mapping is a "chicken-and-egg" problem. In order to localize the device, a pre-existing map of the surrounding is required. However, to build an accurate map of the environment, the technology must know the position and orientation of the device. Smith, Self and Cheeseman [72] first put forward a spatial uncertainty theory which provides a novel idea to solve localization and mapping simultaneously. In the late 2000s, keyframe-based simultaneous localization and mapping (SLAM), introduced by Georg and David [40], allows to get rid of the need for a prior model of the environment by localizing and mapping in parallel.

With the development of artificial intelligence (AI) over the decades, many vision-based robot devices, such as autonomous vehicles, mobile robots, and agents in mobile augmented reality (AR), have arrived in consumer products. These devices often rely on simultaneous localization and mapping to navigate. However, they are commonly operating in dynamic environments, which is one of the main challenges of visual SLAM systems. Matching the scene over subsequent video frames, either direct or through features, allows visual SLAM (vSLAM) to succeed. If an object in the scene moves, and hence the scene changes, the matching constraint becomes unreliable, which causes localization and mapping to start to

drift. The different sensors in SLAM systems, such as monocular cameras, stereo cameras, depth cameras [51], and inertial measurement unit (IMU) systems [86, 43], will influence how well they work in dynamic environments by providing more useful information.

Many different algorithms are published based on different application scenarios to reduce the influence of moving objects. For autonomous vehicles navigation, road detection and marker line detection are popular research topics [35], as the road is always static. For indoor AR systems, the building structure line [101] is employed in the vSLAM system to provide static features. Most man-made buildings follow the Manhattan-world property [101]. Surfaces and the intersecting lines of those surfaces are aligned with three dominant directions. The lines that can be aligned with those dominant directions are defined as structure lines. The structure line can also deal with textureless scenarios, which is problematic for feature-based vSLAM. Some general methods like optical flow [77], K-means clustering [68] and deep learning (DL) [5] are utilized to remove the moving objects.

As AR provides the user with information about the scene, the add-on virtual information sometimes connects to not only a certain position but also a certain type of object. Martin *et al.* provide a good example in their publication, MaskFusion [65] as shown in Figure 1.1. The SLAM systems combined with object segmentation [65, 74, 64] show their strength in this field. QuadricSLAM [55] and CubeSLAM [98] even use objects as the main landmarks.

Our focus is on indoor monocular-based augmented reality system where close dynamic objects often obstruct the camera view. This problem affects SLAM relying on monocular cameras more than systems based on stereo cameras and depth sensors because the absolute

Figure 1.1: (©IEEE 2018) MaskFusion [65]: Using SLAM and object masks in AR. Application shows the use of object recognition and localization to overlay calorie information in an AR application

depth of the scene remains unknown due to a scale ambiguity. The common strategy [97, 100] to deal with dynamic objects in monocular vSLAM is masking out all the moving objects and to only track the background. Tracking the background alone performs better in some cases based on the commonly accepted assumption that background features are always static. However, it reduces the usable image content as dynamic objects cover some of the background. This can lead to failure of localization. The mapping becomes unstable as the number of usable features is not satisfied according to the basic requirements of tracking and optimization. In addition, once a moving object dominates the screen, background features usually locate close to the edges of the screen. Features on edges are less reliable than the features in the centre because of camera distortion, unless the camera is perfectly calibrated, which makes the final estimation unreliable as well.

In order to evaluate the performance of SLAM systems in dynamic environments and the object motion estimation, a test dataset needs to provide not only the ground truth

of the camera trajectory but also the object motion. Our research aims at the scenario that the object temporarily obstructs the camera view with no visible background. However, we are not aware of any public dataset that satisfies our requirement. The SLAM and structure from motion (SFM) datasets only provide the ground truth of the camera trajectory. Although several datasets contain dynamic objects, these moving objects only take up a small portion of the image, which does not influence the estimation too much. Also, we can not evaluate our object motion estimation accuracy as the ground truth of moving objects is not provided. According to this, we decide to generate a novel set of datasets for testing.

1.2 Thesis statements

Monocular feature-based vSLAM suffers from estimation drift in dynamic environments due to the moving features weakening the certainty of image alignment. Once the moving object dominates the image, the estimation of the camera pose becomes unreliable. In this thesis, we aim to solve this problem by tracking background and foreground features separately. If the camera captures enough background features, we estimate the camera pose from it. We also estimate the motion of the moving object from the foreground features. In case the moving object obstructs the camera view and none or not enough background features can be captured, we recover the camera pose from the features on the moving object and the predicted object motion (see Chapter 3 for detail). Experiments are made to show that our system improves accuracy and robustness compared with state-of-the-art SLAM systems in dynamic environments.

1.3 Contributions

We present a new method based on monocular camera to fully exploit all features in each image frame including on dynamic objects. We extend feature-based ORB-SLAM2 [51] to adapt to dynamic environments based on the assumptions that the moving objects in the scene are rigid bodies and that their motion is predictable. These assumptions are often satisfied as if an object is not rigid, we can often treat at least a part of the object as approximately rigid, e.g., the torso of a human. Also, most objects do not move randomly, e.g., because they move purposely or because of momentum. Based on these assumptions, our method makes the following contributions:

1. Based on ORB-SLAM2 [51] for monocular cameras, we present a novel method to estimate camera pose and object motion simultaneously;

2. We present a strategy to recover the camera pose from a rigid object with piecewise constant motion if a moving object obstructs the camera; and

3. We generate a number of new test cases with ground truth for camera trajectory, object motion trajectories, and a semantic segmentation mask for each frame.

Our experimental setup relies on Unity and various public 3D environments: the TUM RGB-D dataset [75], the Replica-Dataset [73], the Matterport3D dataset [13], and self-made datasets for evaluation. In the comparison, our DOE-SLAM outperforms state-of-the-art SLAM systems.

1.4 Thesis structure

This thesis is organized as follows:

- Chapter 2 discusses related work in AR, SLAM and image segmentation. We introduce AR applications in different fields and how AR connects with SLAM. We select various widely used and state-of-the-art SLAM systems and also discusses their performance in dynamic environments. We present the methods used in image segmentation and compare their pros and cons. We also review benchmark datasets that are widely used to evaluate the quality of SLAM systems.

- Chapter 3 introduces the architecture of our method. The flow diagram is shown in that chapter to expound the overall procedure. The major components including object motion detection, and camera pose prediction from object motion are provided in that chapter with a detailed explanation.

- Chapter 4 first introduces the datasets we have utilized for SLAM system evaluation. We provide how we generate the datasets and the assets that we have selected to generate them. We also discuss our assumption and goals in generating these datasets. Then we present the qualitative and quantitative results for our SLAM system. The comparisons with the state-of-the-art methods are provided in this chapter to show our improvement.

- Chapter 5 gives a summary of our work, and the thesis. This chapter also shows the limitations of our work with some recommendations for future directions.

Chapter 2

Related Work

In order to provide a better localization strategy for AR, we present a novel SLAM system to suit dynamic environments. DOE-SLAM is a monocular vSLAM system, that is built upon ORB-SLAM2 with a combination of object segmentation and operation. Although we do not research image segmentation, it is an essential stage in our SLAM system. In this chapter, we first present some AR applications in different fields (in Section 2.1), and the usage of SLAM in AR. Then, we introduce the background of SLAM followed by some state-of-the-art classic SLAM systems (in Section 2.2) as well as dynamic SLAM systems (in Section 2.3). After that, we introduce several popular image segmentation methods (in Section 2.4). To give the background on the evaluation of our SLAM system in a commonly accepted standard, we discuss the SLAM quality evaluation method in Section 2.5.1. Some widely used public datasets for SLAM system testing are presented in Section 2.5.2.

2.1 Augmented reality

Van Krevelen and Poelman [87] introduce and classify AR in their survey. AR contains not only visual displays but also aural displays and haptic displays. For the visual display, AR can be categorized by the devices. Head-worn, hand-held, and spatial visual displays are three popular types of display devices. As a comprehensive problem, AR contains several sub-problems, including display, tracking, and interaction. SLAM is one of the strategies to deal with the tracking problem.

2.1.1 AR applications

AR was first designed for military, medical, and industrial applications. With the development of technologies, AR systems for commercial use and entertainment have appeared soon after. We introduce applications of AR in different fields in the following.

Personal application is one of the biggest potential markets for AR [36]. Gerstweiler *et al.* [27] present a mobile AR navigation system for complex indoor environments. Their system uses some standard mobile devices sensors like inertial sensors and a RGB camera to capture information for tracking. 2D natural features are employed to generate markers to improve the robustness. Wong *et al.* [95] introduce a mobile campus touring system based on AR and GPS to help students easily explore the campus. Archeoguide [89] is an AR-based archaeological sites touring system. It reconstructs ruined sites, and simulates ancient life to help visitors learn the history.

AR is popular in the education area as it provides a clear and intuitive introduction of objects. Bursztyn *et al.* [10] examine the impact of augmented reality field trip exercises

on the interest levels of students using readily accessible mobile devices to prove that AR field trips increase student motivation to pursue geoscience learning. Cai *et al.* [11] introduce an AR system to teach magnetic fields in a junior high school physics course, which explores the effects of using natural interaction on students' physics learning and deep understanding compared to traditional learning tools.

Industrial and military applications are the fields where AR first has been used [87]. In these areas, AR has proven useful in design, assembly, and maintenance. Spacedesign [21] is a Mixed Reality (MR) system for industrial design. The system provides freehand sketching, surfacing and engineering visualization. ARVIKA [23] presents an AR-based vehicle development, production, and service system to simulate collision and assemble products. For the maintenance aspect, Webel *et al.* [93] introduce an AR-based maintenance and assembly tasks training system to improve the efficiency of technicians training. Westerfield *et al.* present an AR with Intelligent Tutoring Systems [94] to assist with training for manual assembly tasks. The system contains a modular software framework for intelligent AR training systems, and a prototype based on the framework that teaches novice users how to assemble a computer motherboard. For industry maintenance, Koch *et al.* [41] introduce a natural marker based AR framework that can digitally support facility maintenance (FM) operators when navigating to the FM item of interest and when actually performing the maintenance and repair actions.

AR is fully exploited in medical applications nowadays. Many AR approaches have been tested in medicine with live overlays of ultrasound, CT, and MR scans [87]. Navab *et al.* [52] design an AR system for an operating room to provide medical intervention. AR

is also used for medical students training, Hamza-Lup *et al.* [32] present a distributed AR system for medical training and simulation. The system presents 3D medical models in real-time at remote locations, which helps students to practice their skills without touching a real patient. Swayze *et al.* present an AR surgical system [78]. The system provides a set of the surgical equipment hardware and the corresponding method to provide an AR display for minimally invasive surgery.

With the development of personal computing devices, AR becomes a hot topic in the personal entertainment field. Pokémon Go [14] is a famous mobile game based on AR, which has over 600 million users in the world. Apple Inc. [56], as one of the biggest consumer electronics companies in the world, equipped their latest tablet (IPAD Pro 2020) with a LiDAR scanner to support AR software better.

2.1.2 SLAM in AR

One of the most fundamental problems currently limiting augmented reality is registration [2]. AR registration, also known as tracking, is categorized by Billinghurst *et al.* in their survey [6]. According to Billinghurst et al., the AR tracking methods can be divided by technology into magnetic tracking, vision based tracking, inertial tracking, GPS tracking, and hybrid tracking.

Magnetic tracking relies on magnetic devices to detect a magnetic field. The transmitter is required to produce alternating magnetic fields as the anchor. The receiver can verify the self pose by measuring the polarization and orientation of the magnetic field. Inertial tracking uses an Inertial Measurement Unit (IMU). Inertial tracking measures three

rotational degrees of orientation based on gravity, and the change in position by using the time period and the inertial velocity. GPS tracking is usually employed in outdoor tracking. The satellite based GPS tracking system allows the GPS device to receive the signal from satellites to localize itself. The current average accuracy is less than 3 meters on smartphone [6], and it requires the satellite signal, so GPS tracking is unsuitable for indoor tracking.

Vision based tracking is popular in AR, as it requires few hardware to support. Mobile devices with the improved computational ability provide platforms for vision based AR system. Vision based tracking can be divided into fiducial tracking and natural feature tracking [6]. Fiducial tracking means we manually add some artificial landmarks into the scene, and using these landmarks for accurate tracking. DeGol *et al.* [16] present an improved structure from motion indoor system by utilizing artificial markers. TagSLAM [58] employs AprilTag [90] fiducial markers for accurate indoor SLAM estimation. Muñoz-Salinas *et al.* [49] improve marker robustness and matching accuracy in the SLAM system. Fiducial tracking handles the problem of textureless areas, areas without structure such as a white wall, and the scene ambiguity problem such as identical same rooms in SLAM. It is usually used in certain indoor environment like a factory, as it provides accurate and robust estimation. However, when we meet a new environment without pre-defined markers, fiducial tracking fails. Natural feature tracking allows the system to track the new environment without any prior knowledge. Natural feature tracking extracts representative features from raw camera image and calculates unique descriptor for each feature. In order to provide an outstanding type of feature, many feature extraction technologies

are published. SIFT [54] (Scale Invariant Feature Transform) is a natural feature detector and descriptor. It allows scale invariant detection of features by using the concept of scale spaces. SIFT is robust and accurate, but it is challenging to run in real-time. SURF or Speeded Up Robust Features [4] is designed to accelerate the procedure while maintaining a comparable level of accuracy. It achieves real-time on many devices but may still be too complex to embed into other systems or devices. Oriented FAST and Rotated BRIEF (ORB) combines the FAST [63] feature detector with the BRIEF [12] descriptor and handles features in a scale invariant and rotation manner.

Model tracking belongs to vision based tracking that estimates the pose by using a known 3D structure. In early works, the 3D model to be tracked was a human-made object [15]. To explore the unknown environment, SLAM appeared as it can simultaneously create and update a map of the real environment while localizing their position within it. PTAM [40] introduces optimization into the SLAM system, which improves accuracy. Later on, many SLAM systems in AR have been published [45, 88].

From 2010, visual based SLAM (vSLAM) technology has been widely adopted in AR applications [80] for tracking. However, there are still some weaknesses, including the initialization problem, pure rotation problem, and scale ambiguity. We will introduce some popular vSLAM systems in the following sections to show their strength and shortcomings.

2.2 SLAM

Simultaneous localization and mapping (SLAM) is a classic problem in robotics and computer vision. SLAM aims to generate the 3D structure of the surrounding and estimate

the self pose at the same time. It allows the robot to the route and build the map. SLAM is inspired by structure from motion (SfM) [57]. SfM, as a computer vision problem, utilizes a collection of 2D images to reconstruct the 3D structure of a stationary scene [57]. SfM aims to reconstruct the 3D model, and SLAM pays more attention on localizing and navigating the device. So SLAM usually requires to operate in real-time, but SfM does not.

Comparing with the SfM where the whole 2D image set is provided in the beginning, SLAM takes new frames during the processing. Thus optimization is significant for SLAM. In early works, researchers adopt filter based methods to optimize the estimation. EKF-SLAM employs extended Kalman filtering [18] to calibrate the measurement. FastSLAM [84] utilizes particle filtering [17] to optimize pose estimation. Later on, some nonlinear optimizations are adopted, such as Levenberg-Marquardt [48], and Gaussian-Newton [9]. Nonlinear optimization can work on the whole trajectory simultaneously, but filter based methods only optimize the current statues as it follows Markov's hypothesis. Based on the property that one frame only shares features with a few frames (compared with the total number of frames), the keyframe correlation matrix is always a sparse matrix, which is cost-less to calculate the Jacob matrix for optimization.

Classic SLAM can be divided into direct methods versus feature-based (indirect) methods [19]. Direct SLAM (e.g., LSD-SLAM [20], DTAM [53]) uses all the information from the sensor without pre-processing. However, feature-based SLAM (e.g., ORB-SLAM [50], PTAM [40]) pre-computes features before feeding them into the actual SLAM threads. Although feature-based SLAM systems contain one more step to pre-compute the meta-

information, the selected features are more stable and less complex than the raw information, which benefits the subsequent SLAM threads. As feature-based SLAM systems only consider representative features, the systems can only build a sparse map that is less readable than a dense map. Although direct SLAM systems can provide a dense 3D map, relying on raw images without pre-computing is arguably less stable due to changes in lighting and viewpoint, and contains more outliers or noise.

Several pre-computation methods have been integrated into vSLAM system, including feature points extraction, semantic segmentation [5], optical flow regularization [77] and depth map prediction [29]. By feature points extraction, representative feature points (e.g., corners) are extracted for subsequent tracking. Once the feature points are extracted, the next step is generating the descriptor for each feature point. Through calculating a well-designed descriptor, the extracted feature points become reliable for matching in various scenarios (including rotation, scale change, light changes.). With the development of machine learning and deep learning, learning-based methods are now commonly used in SLAM. Qiu *et al.* [59] use deep learning to detect loop closure. Tateno *et al.* [81] utilize a convolutional neural network (CNN) to predict depth during pre-processing and for semantic segmentation. They all improve the performance in the part of the SLAM system that they address. However, to the best of our knowledge until now, there is no end-to-end learning-based SLAM system that contains all stages of a typical structure of a SLAM (including camera pose estimation, optimization, loop closure), as SLAM is a complex system which is difficult to solve in a single network.

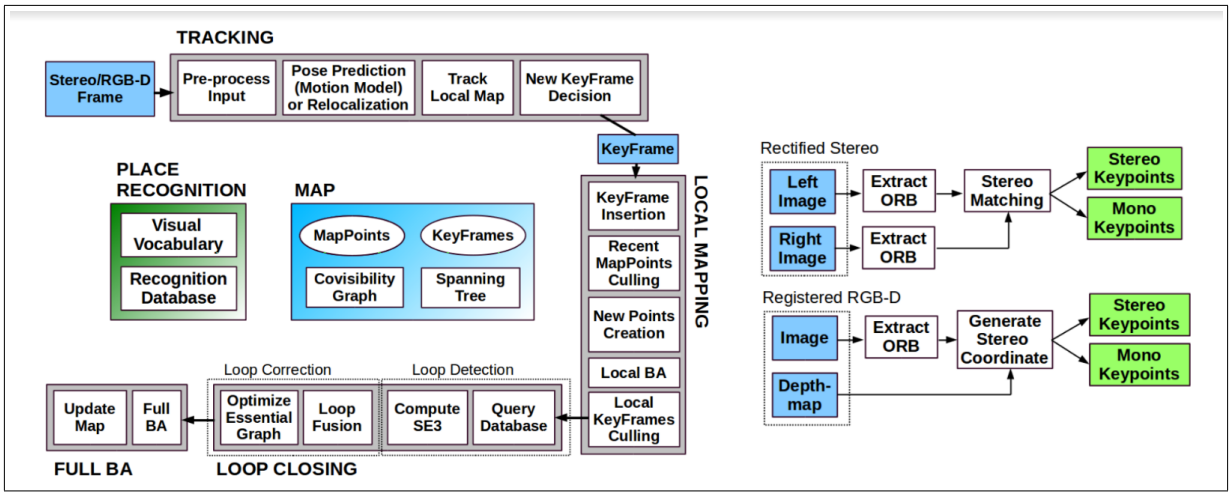


Figure 2.1: (©IEEE 2015) The flow diagram for ORB-SLAM2 [50]

2.2.1 ORB-SLAM2

Mur-Artal *et al.* present ORB-SLAM2 [51] on the top of their previous work ORB-SLAM [50]. As a feature-based SLAM system, ORB-SLAM2 extracts features and calculates descriptors as the pre-processing step. ORB features are employed in all threads of their system. Because even if the feature point is rotated, ORB features still perform well. In addition, the time complexity of ORB is lower than some other popular descriptors, but it still maintains a good matching accuracy. By calculating the pyramid of the image, the ORB feature can also handle the scale change problem. ORB-SLAM2 can not only use a monocular camera as input just like the original ORB-SLAM but also use stereo or depth cameras as input to improve the accuracy by reducing the scale ambiguity.

The whole system contains three parallel threads, tracking, local mapping, and optimization (Figure 2.1 shows the flow diagram of ORB-SLAM2). Although the stereo and depth cameras are used to reduce the scale error, the back-end process for these two types of sensors for tracking and optimization is the same as for the monocular camera. The

system first extracts keypoint features and calculates the corresponding descriptor. After that, the image frame is fed into the first thread, the tracking thread. If the system has initialized, it tries to find enough matches between the keypoints in the current frame and the previous frame by using RANdom SAmple Consensus (RANSAC) [22], then using these matches to estimate the current camera pose related to the previous one from the camera motion prediction or reference keyframe. Otherwise, the system calculates a homography and an essential matrix between two frames to initialize the system.

In order to reduce the time complexity, ORB-SLAM2 only processes keyframes to create new map points and for optimization, because most of the frames provide repeated information, which are not worth to maintain. Bag-of-Word (BOW) is utilized to speed up the ORB descriptor matching. BoW [24] is a visual place recognition method. By building a vocabulary tree and discretizing a binary descriptor space to accelerate the descriptor matching speed. A strategy is employed to delete redundant or invalid keypoints and keyframes to decrease the computing cost so that the number of keyframes would not increase infinitely over time if the scene does not change. ORB-SLAM2 employs two new graph structure concepts, which are covisibility graph, and essential graph to reduce the computing cost of the optimization thread. Otherwise, a large number of keyframes increase the number of edges connecting related keyframes as well, and, the optimization thread would overload the computation. Covisibility graph is designed as an undirected weighted graph. Each node is a keyframe, and an edge between two keyframes exists if they share more than a threshold number of observations of the same map points. The weight of the edge is the number of common map points. Essential graph is the minimum

spanning tree of the covisibility graph.

ORB-SLAM2 optimizes map points and keyframes during its whole runtime. Once the pose of the new frame is roughly estimated in the tracking thread, the system uses Bundle-adjustment (BA) to optimize the local map in order to get a better pose estimate. Bundle-adjustment is the problem of refining a visual reconstruction to produce jointly optimal structure and viewing parameter estimates [85]. The parameter estimates are measured by minimizing some cost functions. The map structure and the camera variations are optimized simultaneously. When the new keyframe is created, the system uses local BA to optimize not only the current keyframe but also the frames connected with the current frame in the covisibility graph. Also, global BA is used to optimize the essential graph once the system detects a loop. It is necessary to keep optimizing the system as the scale of the monocular camera is unknown, and it keeps drifting. Optimization and loop closure are two significant steps that are also widely used to minimize the error and close the scale drift. The pose of map points is refined during the arrival of new keyframes as they would be observed from different views.

ORB-SLAM2 has been tested on various different public datasets to prove its superior performance [50, 51]. However, it also has several weaknesses. The initialization step is demanding. It requires a considerable quantity of keypoints to get a valid initial map. The strategy to estimate the initial pose matrix is by random picking from the homography and the essential matrix. Thus it usually takes a while to initialize the system even in an environment with abundant features. In addition, ORB-SLAM2 relies on feature keypoints with the assumption that most of the keypoints are static and reliable. Thus the perfor-

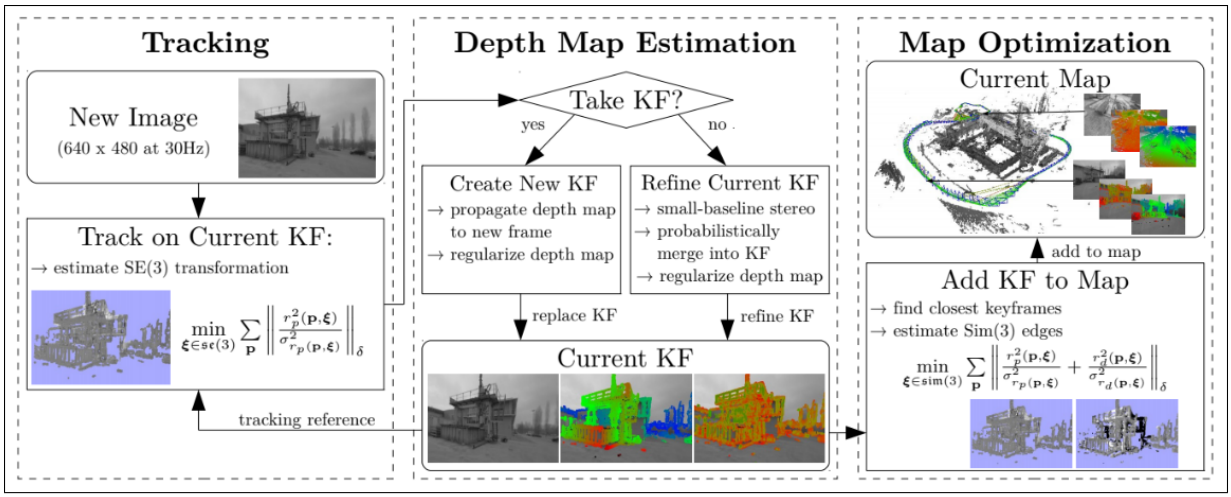


Figure 2.2: (©2014, Springer International Publishing Switzerland)The flow diagram for LSD-SLAM [20]

mance of ORB-SLAM2 in dynamic environments is unstable as the keypoints located on moving objects are also moving, which influences the accuracy, or even worse, breaks the whole system.

2.2.2 LSD-SLAM

Engel *et al.* present LSD-SLAM [20] as a large-scale, direct monocular SLAM system. LSD-SLAM contains three main components, tracking, depth map estimation, and map optimization (see Figure 2.2). It directly estimates the transformation matrix between two frames by a novel scale-aware image alignment algorithm. Their system estimates depth by probabilistically consistent incorporation in tracking to reduce the scale drift.

In the initialization step, the system builds an initial map with a random depth map and large variance from the first keyframe (KF). When a new image comes into the system, the tracking thread estimates the transformation matrix between the current frame and the reference keyframe by minimizing the variance-normalized photometric error. Photometric

error measures the intensity change of two frames upon the assumption that the projections of a point in different frames have the same intensity. Define $I(p, T_i)$ represents the intensity of point p in frame i (with the pose T_i). The photometric error between frame i and frame j is $e_{ij} = \sum_{p \in P}(I(p, T_i) - I(p, T_j))$, P stands for all points in the frame. So the relevant alignment similarity transform matrix between frame i and frame j ($T_{ij} = T_j T_i^{-1}$) can be estimated by minimizing the photometric error e_{ij}.

If the camera moves too far away from the existing map, the system creates a new keyframe from the current frame. Otherwise, the system refines the map according to the current frame. Each created keyframe is added into the map structure for the following optimization thread.

In order to match the same scene at different scales, LSD-SLAM proposes a novel method to perform direct, scale-drift aware image alignment on transformation matrix by not only using the photometric residual e_{ij} like others but also using a depth residual which penalizes deviations in inverse depth between keyframes, to estimate the scaled transformation between them directly. In addition, all keyframes stored in the map are scaled such that its mean inverse depth is one. LSD-SLAM performs well in a large scale environment because all keyframes are optimized at the same time on the same scale.

For loop closure, LSD-SLAM employs an appearance-based mapping algorithm to detect large-scale loop closures. Then the closest keyframes are selected to utilize reciprocal tracking check to avoid insertion of false or falsely tracked loop closures. The system adopts g2o [42] to optimize the map.

A significant limitation of the direct SLAM system lies in the inherent non-convexity of

the image alignment problem [20]. Thus a sufficiently accurate initialization is important. To tackle the image alignment problem in the initialization step, the system uses a small number of keypoints to compute a better initial map. During the loop closure period, the system utilizes two methods to increase the convergence radius, Efficient Second Order Minimization (ESM), and a Coarse-to-Fine Approach.

LSD-SLAM also has some weaknesses. As mentioned in their paper, LSD-SLAM requires a highly accurate initialization. The accuracy of pose estimation decreases if the tracking is affected by a texture-less area or variations in lighting. LSD-SLAM can not deal with dynamic environment cases as objects would influence the image alignment. The direct SLAM system uses all information in the image that may contain outliers and weak information, which affects the robustness and accuracy. For the monocular case, not only LSD-SLAM but also direct SLAM, in general, are sensitive to noise in the frame, because it influences the depth estimation.

2.3 Dynamic SLAM

To improve the accuracy and robustness of SLAM estimation in dynamic environments, many dynamic SLAM systems rely on different sensors, and specific algorithms have been published. Saputra et al. [66] review dynamic SLAM and categorize methods into background/foreground initialization, geometric constraints, optical flow, ego-motion constraints, and deep learning. With the development of deep learning (DL), the deep learning-based approach becomes a popular strategy to solve dynamic SLAM problem (methods [5, 8, 68]). E.g., using deep learning to segment objects and using traditional

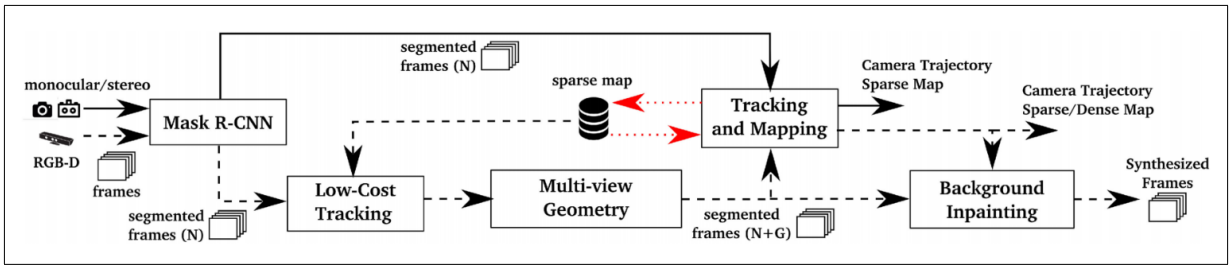


Figure 2.3: (©IEEE 2018) The flow diagram for DynaSLAM [5].

methods to verify each object is static or not. Also, various types of sensors can be added to improve performance in dynamic environments. We discuss some state-of-the-art dynamic SLAM systems in the following sections.

2.3.1 DynaSLAM

Bescos *et al.* introduce DynaSLAM [5] developed from ORB-SLAM2. It can handle monocular, stereo, and RGB-D input with different tracking strategies. In the pre-processing period, DynaSLAM employs Mask R-CNN to segment objects in each frame and only exploits static features to reduce the influence of moving objects.

Figure 2.3 shows the workflow of DynaSLAM. The black continuous lines represent the pipeline for stereo and monocular, and the black dashed lines stand for the RGB-D sensors. The red dotted lines represent the data flow of the stored sparse map. The tracking and mapping block contains the main body of ORB-SLAM2.

Once a new frame comes in, the system first utilizes Mask R-CNN to segment all the *a priori* dynamic content (predefined by authors, e.g., people or vehicles) in the image. Then the branch appears based on the sensor type. Although stereo camera can provide depth information, it can only provide a sparse depth map, which is unstable under their multi-

view geometry method to estimate the object motion. For the monocular and stereo cases, the system directly masks out all dynamic objects segmented in the first step, and only sends the background of the image into the following thread. However, if the input sensor provides depth information, the system utilizes the multi-view geometry to improve the dynamic object segmentation accuracy in two ways. First, they refine the segmentation of the dynamic objects previously obtained by Mask R-CNN. Second, they label as dynamic new object instances that are static most of the time (i.e., detect moving objects that were not set to movable in the segmentation stage).

Camera motion and object motion estimation are intersected. In order to apply multi-view geometry, the camera pose is required as input. DynaSLAM presents a low-cost tracking method to localize the camera pose within the already created scene map by minimizing the reprojection error in the static area. Once the full dynamic object detection and localization of the camera have been done, the system can reconstruct the occluded background of the current frame with static information from previous views.

By using dynamic content segmentation and multi-view geometry, DynaSLAM reduces the effect of dynamic objects as unstable features are not tracked and mapped, and it only relies on static features. It beats many state-of-the-art systems on accuracy. However, weaknesses are also obvious. If the moving object is large enough that features on the background are too few to be tracked, the system gets lost.

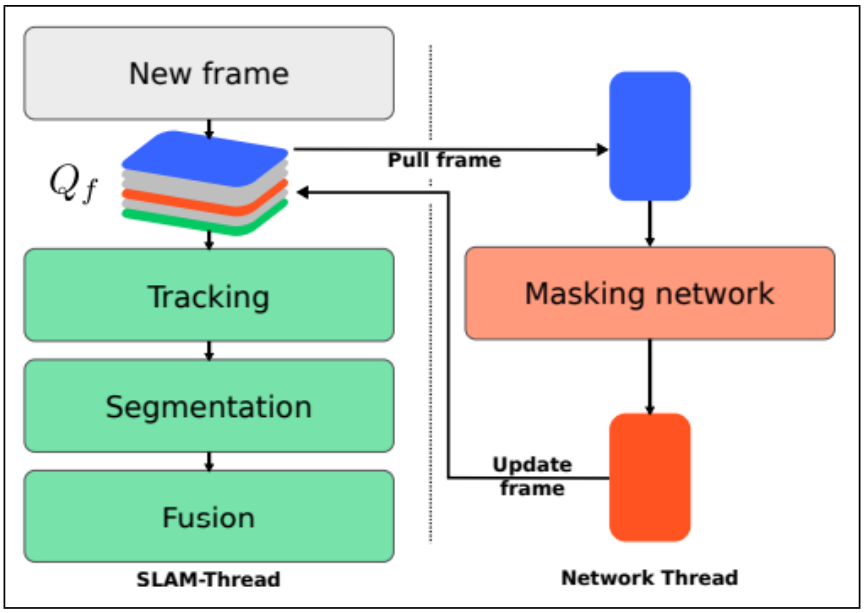


Figure 2.4: (©IEEE 2018) The flow diagram for MaskFusion [65].

2.3.2 MaskFusion

Runz *et al.* present MaskFusion [65] as a RGB-D dynamic multi-model SLAM system. It can not only track and map the static background scene but also estimate object motions and reconstruct object models. MaskFusion takes full advantage of using instance-level semantic segmentation to enable semantic labels to be fused into an object-aware map. It benefits AR systems as camera pose and object motion are both provided, more object-related operations can be implemented.

Figure 2.4 shows the high-level workflow of MaskFusion. The system takes the raw frame from the RGB-D sensor to build a dense map for each object and utilizes Mask R-CNN on RGB data to segment each object and provide the corresponding semantic label in the frame. A geometric segmentation is employed to segment depth data. The following step is tracking and fusion each model separately.

In the tracking step, the system stores each 3D object as a set of surfels. The motion of each object and background is estimated by minimizing an energy that combines a geometric iterative closest point error based on the difference between depth data with a photometric cost based on the RGB brightness constancy between corresponding points in the current frame and the stored 3D model, aligned with the pose in the previous frame. Only the moving objects are tracked separately from the background. MaskFusion takes two strategies to test whether an object is static or not, one based on motion inconsistency, and another that treats objects which are being touched by a person as dynamic.

For the segmentation step, MaskFusion combines two types of cues for segmentation, semantic and geometric cues. Mask R-CNN provides object masks with semantic labels, but it can not be executed at frame rates. So the authors also employ a geometric segmentation algorithm based on an analysis of depth discontinuities and surface normals, which runs in real-time and produces very accurate object boundaries. The geometric segmentation approach follows [82], which generates an edginess-map based on a depth discontinuity term and concavity term. As for the fusion step, the geometry of each object is fused over time by using the object labels to associate surfels with the correct model.

MaskFusion tracks object independently from the background, which decreases the effect of moving objects, as well as builds a model for each object and the background. It fully exploits features in the frame instead of ignoring moving objects. However, the method can not handle the scenario that moving objects cover most of the view, and background features are not enough for tracking. This is despite the fact that MaskFusion takes input from an RGB-D sensor with more features than a monocular camera.

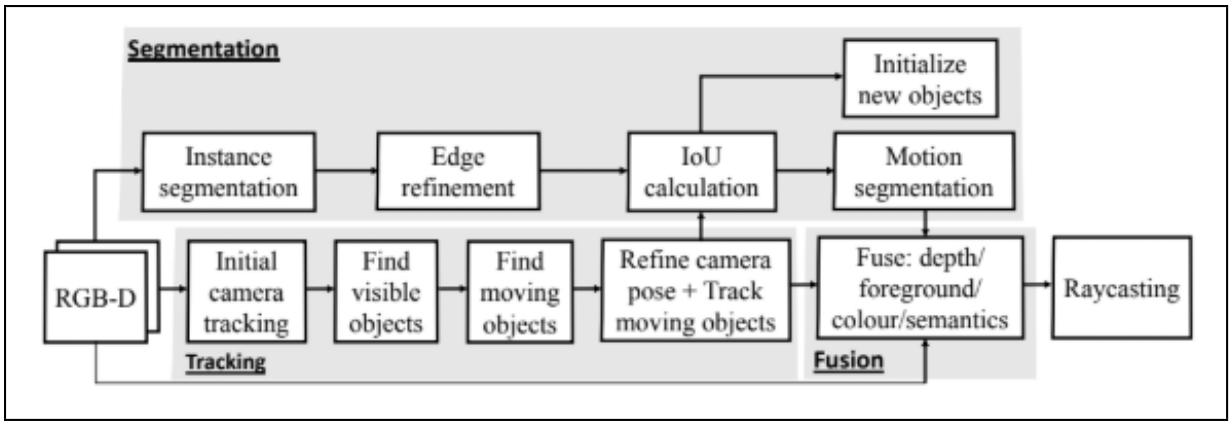


Figure 2.5: (©IEEE 2019) The flow diagram for Mid-Fusion [97].

2.3.3 Mid-Fusion

Mid-Fusion is a multi-instance dynamic RGB-D SLAM system by Xu *et al.* [97]. The system provides robust camera tracking in dynamic environments and, at the same time, continuously estimates geometric, semantic, and motion properties for arbitrary objects in the scene. It creatively adopts an object-level octree to represent objects volumetrically. In an octree, the volume of each object is recursively subdivided into eight cubes with the same size until either the subdivided volume is empty, or the subdivision has reached the leaf level (voxel level). With the octree-based structure, the unused voxels will not be initialized, and the whole system remains memory-efficient [97].

Figure 2.5 shows the pipeline for the Mid-Fusion system. The whole system contains four parts, segmentation, tracking, fusion, and raycasting. As each new RGB-D frame comes into the system, it is processed by Mask R-CNN, which is followed by geometric edge segmentation and the use of motion residuals from tracking to refine mask boundaries.

The tracking is composed of two steps. First, it tracks all model vertices that are stored in the map while masking out detected people; secondly, it tracks against all static

scene parts. Both steps are conducted by minimizing the dense point-to-plane ICP residual and photometric (RGB) residual. After the initial tracking against all model vertices, a threshold is applied to find motion inliers, and the camera pose is refined by only tracking the static scene.

Object pose is also estimated in an object-centric approach, which is less prone to bad initial pose guesses. The system also employs the joint dense ICP and RGB tracking as the camera pose estimation for object tracking. Then the system fuses the detected objects in the frame into the corresponding objects in the map.

The experiment shows Mid-Fusion beats the state-of-the-art dense-tracking SLAM systems in camera localization accuracy. Mid-Fusion also achieves a low object reconstruction error.

2.4 Image segmentation

Image segmentation is the process that divides the digital image into multiple segments. This technology is widely used in object detection. Sivakumar *et al.* review image segmentation techniques [71] and divide methods into two categories, discontinuity-based and similarity-based. Discontinuity-based image segmentation method subdivide images based on abrupt changes in the intensity of gray levels like edge detection based segmentation [38]. Image similarity-based segmentation proceeds by grouping similar pixels depending on the intensity of an image. In this category, images are partitioned into regions that are similar according to a set of predefined constraints. As deep learning has evolved in the past decade, most of the researchers selected to use DL to segment images. Compared

to traditional methods that are typically based on clustering, DL approaches are more accurate and can easily deal with multi-object recognition and segmentation, and can also handle segmentation in various environments.

2.4.1 Semantic segmentation

Semantic segmentation is a popular segmentation strategy to divide the image into different parts based on the object label. Once objects in the image or video are recognized, the machine (e.g., computer, robot.) can understand images and react in a purposed way.

Garcia-Garcia *et al.* present a review of deep neural network-based semantic segmentation methods [25]. In their paper, semantic segmentation methods are divided into region-based semantic segmentation, fully convolutional network-based (FCN-based) semantic segmentation, and weakly supervised segmentation according to the main component of segmentation strategy.

Region-based methods usually first recognize and extract regions from the image and then describe them with semantic labels. Regions with CNN feature (RCNN) [28] are widely used in the region-based category. FCN-based semantic segmentation can be treated as the extension of classical CNN. FCN only contains convolutional and pooling layers without fully connected layer which enables the network to handle input images in different arbitrary size. To solve the problem that manually annotating pixel-wise semantic object masks in a large number of image datasets is quite time-consuming, frustrating, and commercially expensive, some weakly supervised methods have been proposed, which rely on semantic bounding boxes, or even image-level labels.

Many semantic segmentation methods are employed in SLAM system to reduce the influence of dynamic environments. Mid-fusion [97], DynaSLAM [5], and Maskfusion [65] utilize Mask R-CNN [34] to segment objects. DetectFusion [31], and Wang *et al.* [92] select YOLOv3 [60] as the object detection method. Detect-SLAM [100] employs SSD [46] to generate object mask. DS-SLAM [99] uses SegNet [3] to provide semantic label. Code-SLAM [7] adopts U-Net [62] to predict depth information.

Mask R-CNN

Mask R-CNN [34] presented by He *et al.* on the top of Fast R-CNN, is able to generate three outputs for each candidate object, a class label, a bounding-box, and also an object mask. Mask R-CNN performs instance segmentation. Different from the bounding-box, the object mask is a pixel-to-pixel alignment approach that provides a more detailed boundary.

Once an image comes into the Mask R-CNN network, the first stage is utilizing a Region Proposal Network (RPN) to generate candidate object bounding boxes. After objects are recognized, the second step is predicting the class label and the box offset in parallel, which largely simplifies the multi-stage pipeline of original R-CNN. For each Region of Interest (RoI), a Fully Convolutional Network (FCN) is employed to predict the mask, which encodes an input object's spatial layout.

Although Mask R-CNN can provide a detailed boundary instead of only a bounding box, it can hardly run in real-time.

SegNet

SegNet [3] is a deep fully convolutional neural network architecture for semantic pixel-wise segmentation. The structure of SegNet can be divided into an encoder network with a decoder network. The encoder network is inspired by VGG16 network [70], and consists of 13 convolutional layers to classify objects. SegNet discards the fully connected layers in favour of retaining higher resolution feature maps at the deepest encoder output. The decoder network also has 13 layers corresponding to each layer in the encoder network. A multi-class soft-max classifier is employed as the last layer to produce class probabilities for each pixel independently.

As designed, SegNet is significantly smaller and faster than other competing architectures, but still keeps a comparable performance in many tasks.

YOLOv3

You only look once (YOLO) [60] is a state-of-the-art object detection deep learning system presented by Redmon *et al.*. Because of its high accuracy and speed, YOLO has become a popular object detection system.

The detection network has 24 convolutional layers, followed by two fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers.

Although YOLO is accurate and high speed, it still has some limitations. First, the system is weak in the scenario with small objects that appear in groups, As defined, each grid cell only predicts two boxes and can only have one class. Second, it performs poorly

in detecting objects in new or unusual aspect ratios or configurations, because the model is trained according to data.

SSD

Single shot multibox detector (SSD) [46] is a single deep neural network for image object detection. The output space of SSD is discretized into a set of default boxes with corresponding ratios, scales, and locations. SSD utilizes multi-scale feature maps to predict object in multiple scales.

Based on the experiments, SSD can run in real-time with 59 FPS [46], and comparable accuracy with Faster R-CNN [61]. However, some key parameters like minimum and maximum size of default boxes, and the aspect ratio, need to be manually settled for each layer, which can not be learned from training. So the parameter setting highly relies on the users' previous experience.

2.5 Benchmarking

In this section, we introduce some widely accepted error evaluation methods to provide the standard performance measurement of SLAM systems. Some public datasets are also introduced to make test results more convincing.

2.5.1 Evaluation standard

In order to test the performance of SLAM systems, Sturm *et al.* present a series of evaluation methods in their work [76], which are widely accepted and used to evaluate SLAM

systems.

Relative pose error (RPE) represents the local accuracy of the trajectory over a fixed time interval. The RPE measures the drift of the trajectory in a period, which is in particular useful for the evaluation of visual odometry systems. Absolute trajectory error (ATE) measures the global consistency of the estimated trajectory by comparing the absolute distance between the estimated and the ground truth trajectory.

RPE and ATE compare the error on each frame. Root mean squared error (RMSE) as a statistic method is used to calculate the error information on the whole trajectory. Once we estimated RPE or ATE for each frame ($E_{1:n}$). The corresponding RMSE can be calculated as follow:

$$RMSE(E_{1:n}) = (\frac{1}{n}\sum_{i=1}^{n} \parallel trans(E_i) \parallel^2)^{\frac{1}{2}}$$

$trans(E_i)$ represents the translation components of the error matrix E_i. Similarly, we have the RMSE for rotation $RMSE(E_{1:n}) = (\frac{1}{n}\sum_{i=1}^{n} \parallel rot(E_i) \parallel^2)^{\frac{1}{2}}$, $rot(E_i)$ stands for the rotation part of the error matrix. RMSE of ATE is a widely accepted evaluation standard in SLAM. ORB-SLAM, DynaSLAM, and Mid-Fusion utilize this standard to evaluate the performance of their system. EVO [30] is an open source SLAM trajectory evaluation library which can not only measure the RMSE but also draw the visible trajectory map.

2.5.2 Test datasets

Various public datasets have been developed to evaluate SLAM system. Real-world environments sometimes can not provide an appropriate scenario for testing particular scenarios. Thus, another common way to generate test cases is by combining a real scanned

model with a virtual environment to design cases for specific requirements. In the following, we discuss some public SLAM datasets which are recorded in a real-world environment as well as some 3D reconstruction datasets, which can be employed to generate test cases.

Public SLAM Datasets

SLAM datasets are often designed to evaluate specific aspects of SLAM systems. Typically, the respective dataset provides the ground truth of the camera motion. Based on where the dataset has been recorded, SLAM datasets are divided into indoor datasets and outdoor datasets. The outdoor environment usually contains more dynamic objects, like pedestrians and moving vehicles. This type of dataset is typically used in autonomous driving systems. As for indoor environment datasets, most of the objects are static, like desks, bookshelves, and beds. However, the depth of field is small, and some textureless areas like white walls affect the estimation. Indoor datasets are useful in augmented reality (AR) and home robotics.

KITTI Visual Odometry [26] is an outdoor SLAM dataset which contains 22 stereo sequences, and 11 of them provide ground truth. The dataset provides monocular, stereo, laser data, and high accuracy ground truth. In order to generate this dataset, a standard station wagon is equipped with two high-resolution color and grayscale video cameras. Absolute ground truth is provided by a Velodyne laser scanner and a GPS localization system. Figure 2.6 shows two frames of the KITTI dataset.

The TUM RGB-D SLAM Dataset [75] is an indoor SLAM dataset which contains RGB-D data recorded by a Microsoft Kinect sensor, and ground-truth data obtained from

(a)

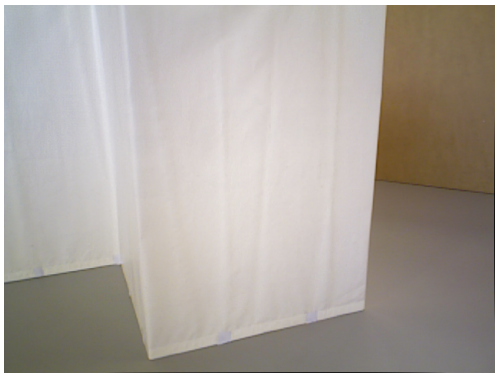
(b)

Figure 2.6: (©IEEE 2012) Samples of KITTI dataset [26].

a high-accuracy motion-capture system with eight high-speed tracking cameras. The TUM dataset as an indoor dataset does not only provide various types of indoor environments, including office scenes and desktops but also provide datasets for scenarios with moving people. In addition, textureless objects such as a white wall may affect feature extraction and reduce the accuracy. The TUM dataset also provides some test cases with structured textureless scenes, such as the white wall with corners in Figure 2.7(c), in order to help researchers develop strategies to overcome this challenge. Figure 2.7 shows some sample images of this dataset.

(a) Static indoor scene

(b) Indoor scene with human moves



(c) Textureless structured scene

(d) Dataset for 3D reconstruction

Figure 2.7: (©IEEE 2012) Samples of TUM dataset [75].

3D Reconstruction Datasets

In some cases, it is hard or costly to generate some special test cases by directly recording. For dynamic environments addressed by our work, we need test cases that provide ground truth for not only camera pose but also for multi-object motions. In this situation, using software to generate virtual datasets is also a good choice. In order to generate virtual datasets close to reality, some 3D reconstruction models can be used to build the background of the scene.

Matterport3D [13] is a large-scale RGB-D dataset containing 10,800 panoramic views from 194,400 RGB-D images of 90 building-scale scenes. The dataset provides annotations

(a) (b)

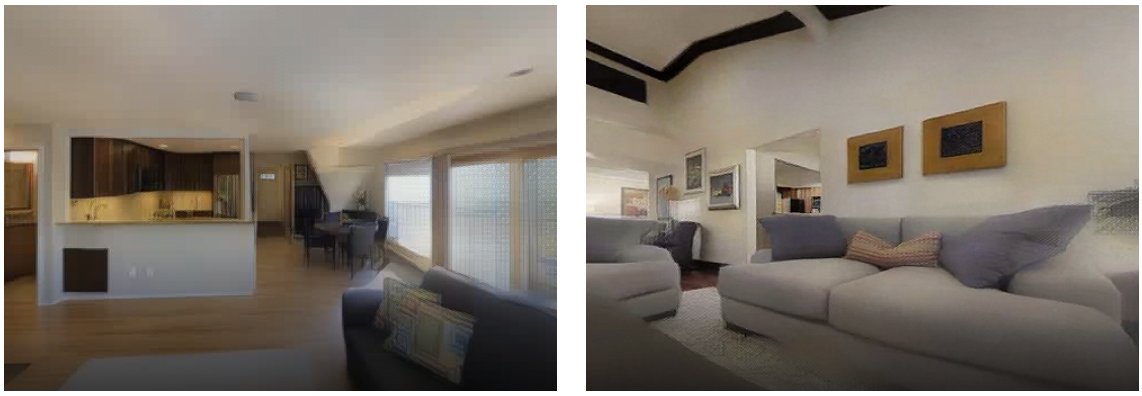
Figure 2.8: (©IEEE 2017) Samples of Matterport3D dataset [13].

(a) (b)

Figure 2.9: (©IEEE 2019) Samples of Replica dataset [67].

with 2D and 3D semantic segmentation, surface reconstructions, and camera poses [13]. Figure 2.8 shows examples from the MatterPort3D dataset.

Replica [73] is another public dataset for indoor environment 3D reconstruction. For each reconstructed scene, it contains clean, dense geometry, high resolution, and high dynamic range textures, planar segmentation as well as semantic class and instance segmentation. Replica contains various indoor environments like apartments, offices, single rooms. The dataset also provides glass and mirror surface information, which makes the scenes more realistic. Figure 2.9 shows samples of Replica dataset.



(a) (b)

Figure 2.10: (©IEEE 2018) Samples of Gibson Environment dataset [96].

Gibson Environment Dataset [96] is an indoor space dataset. It contains 572 models and 1440 floors from various environments including households, offices, hotels, venues, museums, hospitals, construction sites. For each space, the dataset provides the 3D reconstruction, RGB images, depth, surface normal, and semantic object annotations. The dataset is captured in panoramas which contains more information in each frame, but requires one more step to deal with the panorama image. Figure 2.10 shows samples of Gibson Environment Datasat.

Stanford 2D-3D-Semantics Dataset [1] provides a variety of mutually registered modalities from 2D, 2.5D and 3D domains, with instance-level semantic and geometric annotations. The dataset covers over 6,000 m^2 and contains over 70,000 RGB images. The dataset also provides the depths, surface normals, semantic annotations, and the registered raw and semantically annotated 3D meshes and point clouds.

2.6 Summary

In this chapter, we first introduce the development and applications of AR with how SLAM works in the AR system. We then discuss several classic and some dynamic environment SLAM systems. Some deep learning networks embedded in dynamic SLAM systems are summarized. We also discuss benchmarks involving standard datasets as well as accepted measures to judge the efficacy of a method.

In the following chapters, we first introduce the methodology of our dynamic object enhanced SLAM system, followed by the experiments to show our improvements compared with selected state-of-the-art technologies. A conclusion is made in the end to discuss the performance and limitation of our work. Some future work directions are provided.

Chapter 3

Dynamic object enhanced SLAM

In this chapter, we introduce our work, a monocular vSLAM, which performs well in dynamic environments. As presented in Chapter 2, dynamic environments are one of the main challenges in monocular vSLAM. The absolute depth is unknown, and moving objects influence the image frame alignment and finally affect the accuracy. Many researchers solve this problem by adding more types of sensors like IMU, depth cameras, and radar to capture the absolute scale. Deep learning has been adopted to tackle this problem by segmenting image frames in the pre-computing period. However, due to the limitation of the monocular camera, the most common strategy to avoid effects from moving objects is ignoring moving features and only rely on static background features. Ignoring moving objects improves the ability of SLAM systems to deal with a dynamic environment but this strategy weakens the robustness as the number of useful features decreases. In order to solve SLAM in dynamic environment, we present a novel method that not only tracks from background static features but also tracks moving objects separately. Once an object covers most of the camera view and the system can not capture enough background features

for tracking, we can predict the camera pose from the foreground object.

This chapter contains five sections covering the major aspects of our system. We first show the overview structure and explain the workflow. Then we present how our method constructs the object model separately from the background, as well as present our strategy to approximately track the object and estimate the object motion. Next, we introduce the method to recover the camera pose from object motion when the background features are lost. A summary is presented at the end of this chapter to highlight the main features of our system.

3.1 Overview of the system

ORB-SLAM2 is widely used and tested on many public datasets. Also, it has been proven that ORB-SLAM2 is stable, highly accurate, and performs well in both indoor and outdoor environments. We consider ORB-SLAM2 suitable as the foundation of our work. Nevertheless, in highly dynamic environments, traditional ORB-SLAM2 becomes unstable as moving features affect image alignment, which leads to pose estimation drift.

We build our DOE-SLAM method upon ORB-SLAM2 to operate successfully in dynamic environments. We use a monocular camera as the only sensor motivated by the fact that mono cameras are cheap and readily available in AR systems. We make modifications mainly in tracking and local mapping for which the flow diagram of DOE-SLAM is shown in Figure 3.1. The flow diagram shows the case of at least one detected foreground object in the current frame. For static scenes, DOE-SLAM works precisely the same as ORB-SLAM2. If there are multiply moving objects in the frame, we current only track the

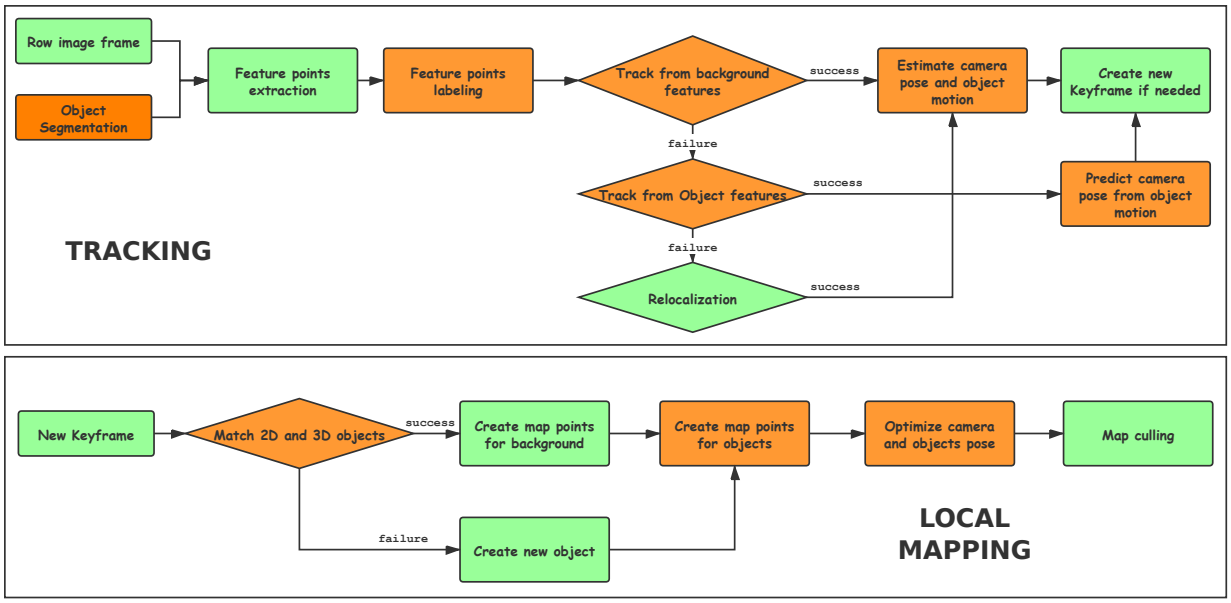


Figure 3.1: Tracking (top) and local mapping (bottom) of DOE-SLAM. The changes compared to ORB-SLAM2 are shown with orange boxes.

largest one, as it has the highest probability to cover the camera.

We provide an interface to connect a deep learning-based object detection method to generate object segmentation information. Mask R-CNN is pre-compiled in our system as the pre-processing step for testing. However, our work does not focus on object detection, the main body of our system accepts images from a monocular camera and their corresponding object segmentation masks as input. In this way the user can adopt and switch to their preferred object segmentation method easily. DOE-SLAM uses the same three threads as the basic structure of the system as ORB-SLAM2: tracking, local mapping, and loop closure, all three running in parallel.

A new data structure named *spcObject* is introduced into our system to store the sparse point cloud model for each object. Each *spcObject* class only contains the reference to all map points belonging to it, so that the *spcObject* class would not increase the storage usage

of the system too much. Different from ORB-SLAM2 that map points directly belonging to the map, in our DOE-SLAM, map points first belong to the corresponding object, and objects belong to the map. By doing so, our system can operate at the object level. Our system stores the motion of each object in each frame. The object pose is also stored. The initial camera pose is set to be the identity matrix, which means the object shares the coordinate system with the background map. A new object is created once the camera captures it, and the system releases and deletes the object from the map when the object moves out of the camera view over a number of frames.

Tracking is the main contribution of DOE-SLAM. The input to DOE-SLAM is a sequence of images from a mono camera and their corresponding segmented object masks. We tested Mask R-CNN [34] as well as manually labeling to provide a detailed segmentation mask. In the feature extraction step, our method gives each feature point a class id marking if it belongs to a foreground object or the background. Then, we use background features to estimate the camera pose. If tracking succeeds, we also estimate the object pose (see Section 3.3 for details). Otherwise, we can predict the camera pose from the object pose if there is a tracked object in the scene (Section 3.5).

The primary tracking method is the same as ORB-SLAM2. It is a 2D-3D scene alignment problem. We need to find out the best camera pose to minimize the projection error. To match the 2D features in the current frame and the 3D points in the created map, the system first tries to find enough matching 2D feature points between the current frame and the last frame. Then based on the matches between 2D feature points and 3D map points in the last frame, we build connections between 2D points in the current frame and the 3D

map points. Once enough inlier matches are found, our system utilizes perspective-n-point (PNP) [44] to estimate the camera pose for the current frame. For object motion estimation, we use the object motion estimated in the last frame as an initial guess for the current frame. Then, we utilize local bundle adjustment (BA) only for the features on the object to optimize the motion in the current frame to get an accurate estimation. We assume that when the object first appears in the camera view, its size is small enough such that it is not obstructing the complete view. We believe this to be a reasonable assumption because an object would not suddenly appear in full view but would likely gradually come into full view. However, if the size of an object is too small, there are not enough features for object motion estimation, as we need at least eight points for the estimation [33] and many more points for optimization. If the object is too small, the number of feature points is small, which leads to an unreliable object motion estimation. We set up a threshold for the object size in a frame, in order to decide if an object is worthwhile to track. ORB-SLAM2 utilizes two tracking strategies to localize the camera, named tracking from motion model, and tracking from reference keyframe. For each new frame that comes in, ORB-SLAM2 first checks whether the camera follows the same motion model as in the previous frame. If not, the system tries to estimate the camera pose by aligning the current frame with the reference keyframe. If these two tracking strategies both fail, the system considers the camera is lost. In DOE-SLAM, we present a new tracking method to track from a moving object. If the system is not able to track from static features, we try to predict the object motion from the object motion recorded in previous frames and recover the camera pose by tracking the moving object.

The local mapping thread in ORB-SLAM2 creates new map points, optimizes camera pose by local BA, and culls the map structure. It is a thread to maintain all types of data in the system. We add additional steps to deal with object point cloud modeling and object motion estimation. Once a new keyframe comes in, we first find matches between objects in that keyframe (2D features) and objects in the map (3D map points). If enough matches are found, we create new map points and add them to the matched objects. Otherwise, the system creates a new object into the map. Once the map points are generated, we optimize the camera and the object pose separately if the object is moving.

Loop closure is a significant step for accurate maps. However, incorrect loop detection and closure can reduce the accuracy below that of a system without loop closure. Considering the importance of loop detection, we only use static background features to detect the loop. As our final goal is estimating the camera trajectory, our system ignores map points belonging to objects in the loop correction step, and utilizes the global BA to optimize all keyframes and background map points.

3.2 Object modeling

As the object may move but the background is always static, it is necessary to separate object features from background features. Given an image frame and its corresponding segmentation mask, we first extract all ORB feature points. Our system also treats the background as an object. Once we get all features in the frame, the system estimates the camera pose only based on background features. If the estimation succeeds, the next step is to find the set of matched feature points pair Γ in two frames based on ORB feature

matching. We assume that each feature point in frame i can only match one feature point in frame j. Let o_i^a denote the a^{th} object in frame i and p_i^x represent the x^{th} feature point in the frame i. Define the function $c(p)$ to return the object which feature point p belongs to, $c(p_i^x) = o_i^a$ if p_i^x belongs to o_i^a. Given frame f_i, frame f_j and $\gamma = (p_i^x, p_j^y)$ representing the element in Γ, we have the following method to match the object:

$$BId(o_i^a, f_j) = \underset{b}{argmax} \sum_{\gamma \in \Gamma} match(\gamma, a, b) \tag{3.1}$$

$$match(\gamma, a, b) = \begin{cases} 1, & if\ c(p_i^x) = o_i^a\ \wedge\ c(p_j^y) = o_j^b \mid \gamma = (p_i^x, p_j^y) \\ 0, & otherwise \end{cases}$$

For each object o_i^a, our system finds the best matching object o_j^b in f_j by calculating $BId(o_i^a, f_j)$. If $BId(o_i^a, f_j) > \tau$, we consider that o_i^a matches o_j^b. We define the threshold τ as three-quarters of the number of feature points belonging to o_j^b in f_j.

After the matched 2D object pairs are found, the pose of the corresponding object in the current frame can be measured (see Section 3.3 for detail.). Moreover, the system can establish a sparse point cloud for each detected object by giving each map point a class id.

In addition, we also need to separate 3D map points belonging to objects and belonging to the background. So once the map has been updated, we need to match the 2D object with the 3D object. The feature point's id in each keyframe is independent, but the 3D object id is unified through the whole process. For each keyframe, we correlate the feature point's class id with the object class id to find the match between the 2D object in the image and the 3D object point cloud in the map. Figure 3.2 shows the result of our object

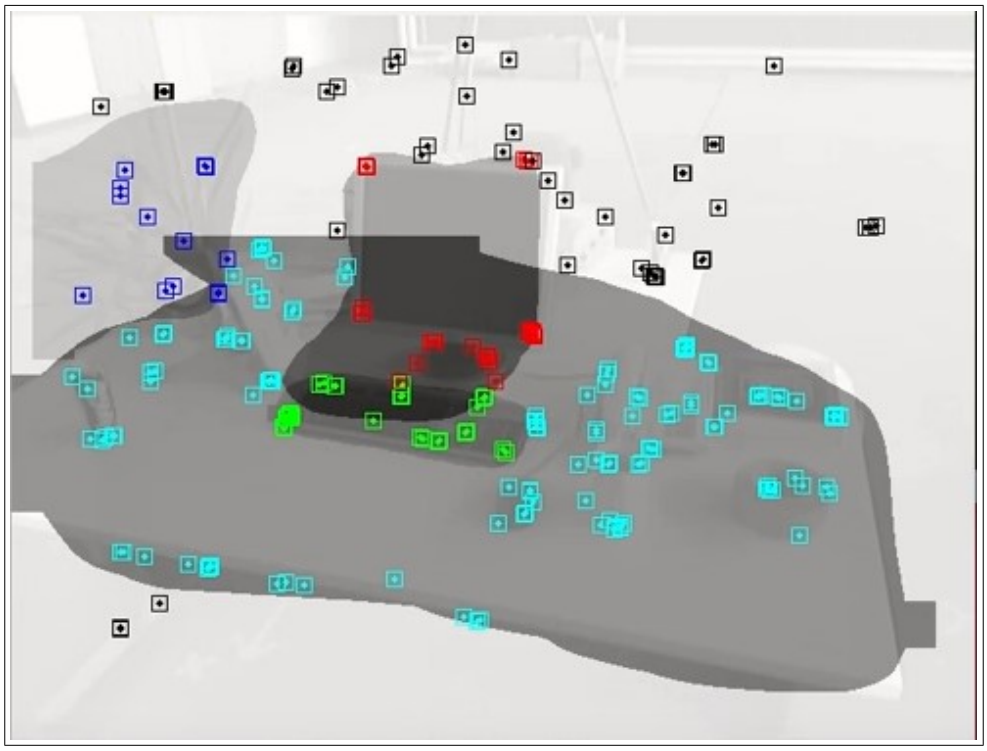


Figure 3.2: Object modeling based on segmentation.

modeling method during a test on the TUM RGB-D dataset [75] with Mask R-CNN [34] to segment the image. There are five objects in the image: monitor (red), keyboard (green), plant (blue), desk (cyan-blue), and the background (black).

It is notable that the masks from object detection may overlap with each other (as in Figure 3.2, the keyboard overlaps the desk). To tackle this problem, we first sort the object masks by size, then for each feature, find the smallest mask it belongs to and assign the feature point to that object. The smaller object is likely on top of the larger object, as otherwise, the small object can not be seen and detected. If the larger object partially obstructs the smaller object, the covered part of the smaller object can not be detected.

3.3 Object motion estimation

Different from other SLAM methods, we not only track the camera pose but also estimate the motion of a moving object. In the SLAM problem, the final goal is to estimate the transformation matrix T_w^c between the world and the camera coordinate systems. However, in dynamic environments, the motion uncertainty of the feature points influences the estimation. Object motion estimation interconnects with camera pose estimation, which means we can estimate T_w^c and the relative motion between the object and the camera T_o^c, but we can not estimate the object motion in the world frame T_w^o directly. Instead, we estimate T_w^c from background features ignoring the effect of object features first. Then, we measure T_o^c relying only on object features and calculate the object-world transformation matrix $T_w^o = T_c^o \; T_w^c = (T_o^c)^{-1} \; T_w^c$. For each frame f_i, we estimate ${}^iT_w^o$ in the first step. The motion of the object from frame f_j to f_i can be calculated as ${}^{j\to i}T_w^o = {}^i T_w^o \; ({}^jT_w^o)^{-1}$.

We define the coordinate system of the object in the same location as the world coordinate system in the beginning. If the object is static from the beginning to the frame f_i, we have the equation ${}^iT_o^c = {}^i T_w^c$. Similarly, if the object is static from f_i to f_j, the motion of the object satisfies ${}^{i\to j}T_w^o = I$ (the identity matrix). The object motion we estimated in this step can be treated as an initial estimation. A more accurate motion estimation result will be produced after the object motion optimization step based on this initial estimation (see Section 3.4 for detail).

We set up a rule that our system only captures objects with a size over a certain ratio of the image frame size. This rule is needed because if the object is small, the number of feature points on that object is also small and the motion estimation is unreliable.

We select to use the object size instead of the number of features on an object for the threshold because we can easily get the size of the object from the object segmentation at the beginning but the number of features can only be obtained after the feature extraction. Using object size can speed up the process.

3.4 Object motion optimization

Optimization is an important step to minimize the scale drift between each frame. As the absolute scale of the environment is unknown in the monocular camera case, what ORB-SLAM does is manually set up a scale factor for all features and keep the factor unified through the whole process. In other words, the system does not know the exact scale but keeps the whole environment on the same predefined scale, which usually differs from the real scale. If we optimize T_o^c and T_w^c separately from the beginning, the scale difference between background and objects will increase over time. In order to lower the scale difference, we separate the optimization only if the object moves in the local map. Otherwise, we keep optimizing foreground object points and background points together to unify the scale.

By roughly estimating T_o^c and T_w^c through separating features on the background and objects, the initial object motion can be calculated. Our system utilizes hysteresis thresholding to detect whether the object is moving based on the initial estimation from Section 3.3. To improve the accuracy of motion verification, we employ two thresholds according to the object motion in previous frames, τ_{start} to detect when the object starts moving (the object is static in the last frame), and $\tau_{maintain}$ to test if the object keeps moving (the object

is moving in the last frame). We estimate the object motion with $mScore_t$ for translation and $mScore_r$ for rotation separately. We consider the corresponding translation submatrix t and rotation submatrix r of T separately.

$$mScore_t = \|^{i \to j}\ t_w^o\ \|_2; \quad mScore_r = \|^{i \to j}\ r_w^o - I\ \|_2 \tag{3.2}$$

$$isRotate(^{i \to j} r_w^o) = \begin{cases} True, if\ isMoving(^i T_w^o)\ and\ mScore_r > \tau_{rot\_main} \\ True, if\ not(isMoving(^i T_w^o))\ and\ mScore_r > \tau_{rot\_start} \\ False, otherwise \end{cases}$$

$$isTrans(^{i \to j} t_w^o) = \begin{cases} True, if\ isMoving(^i T_w^o)\ and\ mScore_t > \tau_{tran\_main} \\ True, if\ not(isMoving(^i T_w^o))\ and\ mScore_t > \tau_{tran\_start} \\ False, otherwise \end{cases}$$

$$isMoving(^j T_w^o) = isTrans(^j t_w^o)\ or\ isRotate(^j r_w^o)$$

The thresholds are obtained heuristically. We verify the object motion status based on not only the current frame but also the previous frame, which improves the accuracy and the robustness of our system compared with other systems only using the current frame for estimation. Only if an object is detected to be dynamic in three frames continuously, our system marks the object as moving. Once we verify the object motion, the backend of

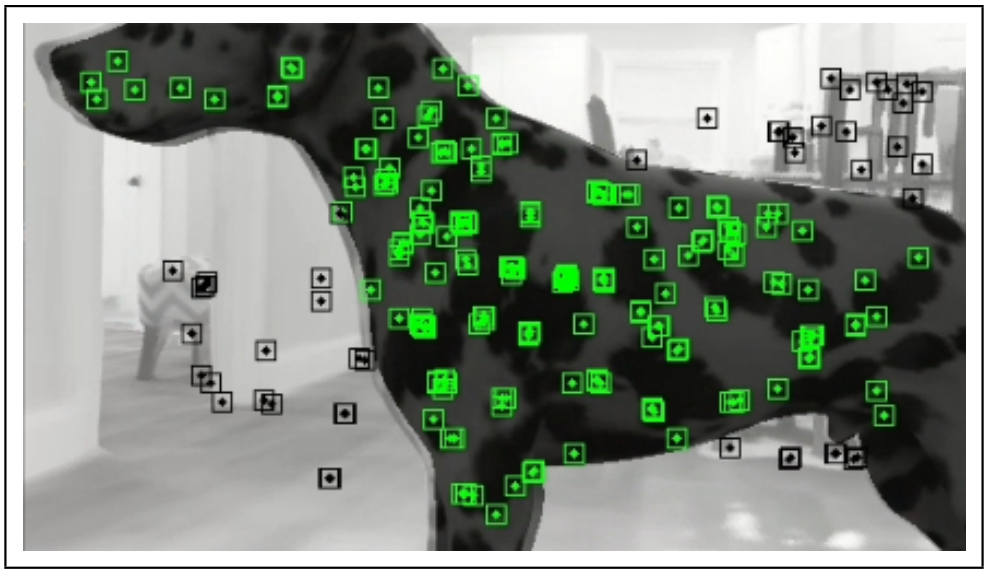


Figure 3.3: A moving object (dalmatian dog) covers most of a frame.

our system will optimize the camera pose and the object pose separately by using BA if moving objects are detected.

It is difficult once an object moves out of view, to verify if a newly detected object is the identical object or another object which looks the same. Thus we set up a strategy with a threshold that if an object moves out of the screen and can not be detected over a number of frames, we delete the object model from the map so that it can not be matched with newly detected objects. As a consequence, once an object moves out and then moves back in, we create a new model for it instead of using the previous one. This not only reduces storage usage but also increases the robustness.

3.5 Camera pose prediction from object motion

Our strategy for tracking mainly relies on background features in order to reduce the effect of moving objects. However in the scenario shown in Figure 3.3, the camera can not capture

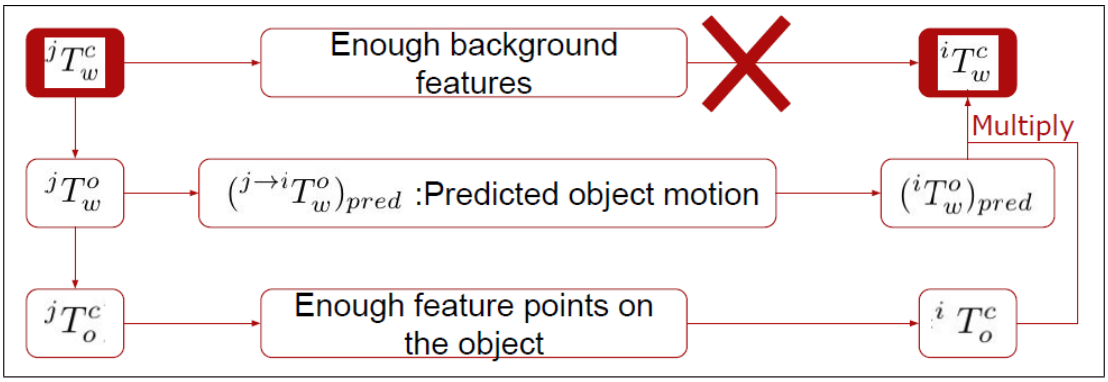


Figure 3.4: The flow chart for using the moving object to predict the camera pose when there are not enough background features.

enough background features, which causes the background-camera pose estimation in the current frame to fail, or even worse, the mapping to fail. In contrast, the features on the object are plentiful for object-camera pose T_o^c estimation. To overcome this problem, we designed a dynamic object enhanced camera pose prediction method to predict the camera pose from the object motion. The first step is predicting the object motion $(T_w^o)_{pred}$ from the object motion model recorded in the previous frame. In this case, the object size in the frame before background tracking gets lost is large enough for object motion estimation, and the result is reliable so that our camera pose prediction is also accurate (see our experiments in Chapter 4 for details).

Figure 3.4 shows our strategy to predict the camera pose from the moving object. If the system records a series of object motions before background tracking gets lost, we can exploit the stored information to fit the object motion model and predict the following motion. We tried several methods to predict motion, including an Extended Kalman Filtering [37] and constant motion prediction. Other prediction methods could also be adopted in future work. We finally selected constant motion prediction in our system. The

object motion estimated in the latest frame is likely to be more accurate and useful than in previous frame. It is reasonable because when an object first appears in the camera view, the size at which the object is captured is likely small, and the number of feature points are also small, so the feature based tracking for the object is unstable and unreliable. With more frames where the object is visible, more feature points are added. Also, only if the size of the detected object becomes large it will obstruct the background but then, the motion of the object can be estimated more accurately. The object motion may vary over time, the object motion estimated in the last frame is likely closer to the current object motion status than the motion estimated in previous frame. As our system's main goal is camera localization, we care more about the object motion between every two frames than the absolute object position. For this camera localization task constant motion prediction performs better on our test results. We assume that acceleration is small between two frames, and hence we can use a constant velocity motion model to predict the object motion. Given that the frame rate of the test video is 30 fps, and the time interval between every two frames is less than 34ms, acceleration can be ignored for many objects in such a small time interval. According to this assumption, our system estimates the camera pose in f_i from f_j as follows:

$$\begin{aligned} {}^{i}T_{w}^{c} &= {}^{j\to i}\,T_{o}^{c}\,\left({}^{j\to i}T_{w}^{o}\right)\,{}^{j}T_{w}^{c} \\ &= {}^{i}T_{o}^{c}\,\left({}^{j}T_{o}^{c}\right)^{-1}\,\left(\left({}^{i}T_{w}^{o}\right)_{pred}\right)\,\left({}^{j}T_{w}^{o}\right)^{-1}\,{}^{j}T_{w}^{c} \end{aligned} \tag{3.3}$$

The potential error in our prediction method will accumulate with time. However, in most scenarios, it is reasonable that the method improves localization. On the one hand, if

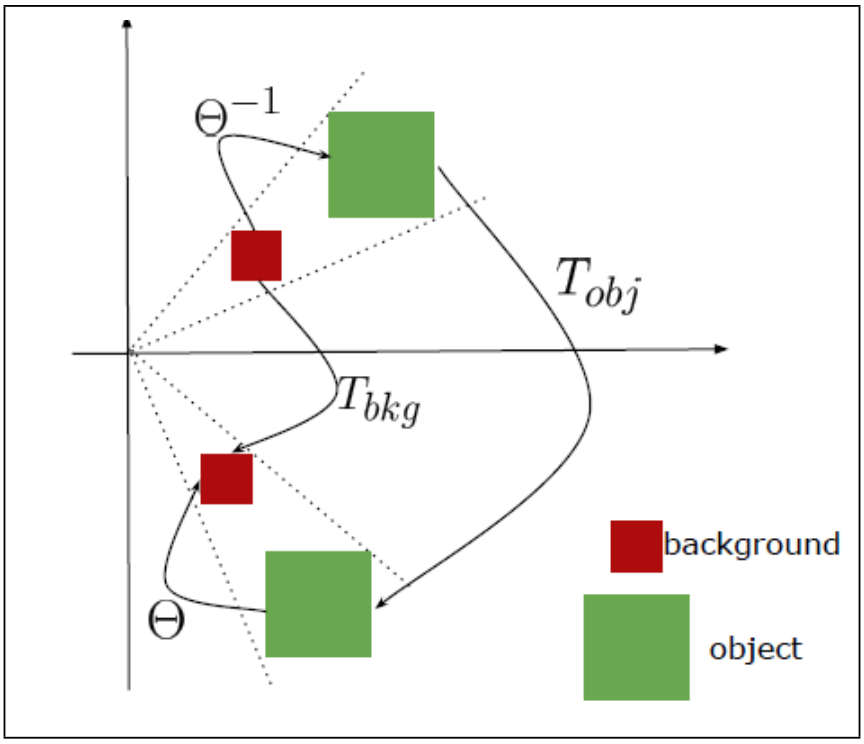


Figure 3.5: The graph to show the scale difference.

the object moves fast, there will be only a few frames with not enough background features where we have to rely on object motion. On the other hand, if the object moves slowly, the object motion will not influence the estimation too much as the object is nearly static.

As the model of the moving object is built separately from the static scene, the object may utilize a different scale than the background. As a monocular camera can not capture the real depth of the scene, the system sets the scale by keeping the average inverse depth of all map points to 1. In SLAM, the core problem is localizing the camera, and we will show next how the scale difference affects the camera pose estimation. Without loss of generality let the scale difference between background and foreground moving object be $\Theta = S_{obj}S_{bkg}^{-1}$ (S_{obj} represents the scale matrix of object, S_{bkg} represents the scale matrix of background). The object motion ${}^{j\to i}T_w^o$ estimated in Section 3.3 is actually based on S_{obj}.

The object motion shares the same scale with the background is $\Theta\ {}^{j\to i}T_w^o\Theta^{-1}$. Figure 3.5 shows how the scale difference affects our prediction. Considering the scale difference Θ, we can rewrite the previous prediction Equation 3.3 as follow:

$$\begin{aligned} {}^{j}T_w^c &= (\Theta^{i\to j}T_o^c\Theta^{-1})\ (\Theta\ {}^{i\to j}T_w^o\Theta^{-1})\ {}^{i}T_w^c \\ &= \Theta^{i\to j}T_o^c\ {}^{i\to j}T_w^o\Theta^{-1}\ {}^{i}T_w^c \end{aligned} \tag{3.4}$$

The equation above shows how scale difference between the object and the background affects the accuracy of our prediction. For simplicity, we did not mention the scale similarity transform in the previous sections. The camera pose we predicted when an object covers the camera would not affect the following camera pose estimation once the object moves away. When the camera captures the background scene that it has visited before, the system automatically optimizes the scale difference. Moreover, the system utilizes the camera motion estimated in the last frame to provide an initial camera pose in the current frame. It also optimizes the scale difference when the system switches to track based on the object instead of the background.

3.6 Summary

In this chapter, we discuss the strategy in the design of our SLAM system to overcome the problem of a dynamic object obstructing the camera view. In tracking, we not only track the camera pose but also estimate the motion of moving objects. If the moving object eventually blocks the camera view completely, our system is able to estimate the camera pose from the prediction of object motion. We also argue that even if we are mapping

the moving object and the background separately in a mono-camera scenario, the scale difference between them will not affect our estimation too much.

In the following chapter, we introduce the tools we utilize to generate test cases. Then we discuss the overview of the datasets we have selected and how we generate data for testing. The detailed explanation of our experiments is also shown in the next chapter.

Chapter 4

Experiments

In this chapter, we measure DOE-SLAM in several different scenarios to show the performance in accuracy and robustness. We select ORB-SLAM2 and DynaSLAM for comparison. As introduced in Chapter 2, ORB-SLAM2 and DynaSLAM are two state-of-the-art vSLAM systems. ORB-SLAM2 is a monocular vSLAM system which is the foundation of our system, and DynaSLAM aims to deal with the problem that the scene contains moving objects. DynaSLAM is able to work with a monocular camera. Although many SLAM systems that focus on dynamic environments have been published (discussed in Chapter 2), their systems are mostly based on other types of sensor such as a depth camera and IMU. A number of them consider the monocular case, but the strategies to avoid the influence of moving objects are more or less the same.

Our system estimates not only the camera pose but also the object motion, thus the dataset for testing also needs to provide the ground truth for object motion. We generate test cases, as shown in Section 4.1. Each test case contains a sequence of image frames, the corresponding segmentation mask for each frame, and the ground truth for both the

camera trajectory and the object motion.

Our experiments divide into three parts, in Section 4.2 we consider the situation that the object in the view is static at first and then starts to move. In Section 4.3, the object passes through the view without stopping. We also test our system on selected TUM datasets in Section 4.4 to have a standard reference point. As introduced in Section 2.5.1 Root-mean-square error (RMSE [76]) for absolute trajectory error (ATE) is calculated to evaluate the accuracy, and the number of frames when tracking is lost is counted to evaluate the robustness. All experiments are tested on an Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz with 16 GB RAM running Ubuntu 16.04. Our test result graphs share the same legend (such as shown in Figure 4.2).

4.1 Dataset generation

In this section, we introduce the tools and the datasets we use to generate the test cases. To evaluate DOE-SLAM in depth as well as to show the progress of DOE-SLAM, we utilized two types of datasets for testing. Section 4.1.2 shows the test cases we generated from several 3D reconstruction datasets. Section 4.1.3 presents the selected test cases from the public TUM dataset.

4.1.1 Unity3D

Unity3D is a popular 3D game engine, which is particularly suitable for independent game developers and small teams [91]. Unity3D markets itself as "a feature-rich, fully integrated development engine that provides out-of-the-box functionality for the creation of interactive

3D content". Complete toolset, intuitive workspace, and editing feature of Unity allow developers to save time and effort [39].

We employ Unity3D to generate test cases. Compared with recording video directly, generating a virtual environment allows the user to get the precise ground truth without any expensive motion detection device. It is difficult to detect and control the ground truth of the object motion in the real world, but in software simulation, we are able to design the object motion as desired, and the ground truth of the object motion is known.

To generate test cases on the Unity3D engine, background scene models are necessary. We use three sources of background models in our simulations: virtual environments from Unity3D asset store [69], the Replica dataset, and the Matterport3D dataset. As discussed in Section 2.5.2, the Replica and Matterport3D datasets are two 3D reconstruction datasets. The dataset is realistic as the 3D models of the scene are constructed from real-world recordings. Unity3D supports the model in a mesh format. However, the Replica dataset only consists of point cloud format model, which can not be imported into Unity3D directly. Pcx, point cloud importer/renderer for Unity3D [79], is a third-party tool, and allows the user to import and render point clouds in Unity3D.

4.1.2 Test case generation

To evaluate the performance of our system, we require the test case to provide an image sequence, the corresponding segmentation mask sequence, the ground truth for the camera trajectory and for the object motion. Based on these requirements, we generated seven different test cases in total based on three different background scenes. The detailed

Table 4.1: OVERVIEW OF GENERATED TEST CASES

		Background Model	Frame number & Video length (s)	Frames with Moving Object
Previously Static	Test case 1	Unity3D Assert	1974(62)	1840 (93.21%)
	Test case 2	Replica	1596(20)	1080 (67.67%)
	Test case 3	Matterport3D	2385(60)	994 (41.67%)
Fully Dynamic	Test case 4	Replica	614(28)	320 (52.12%)
	Test case 5	Matterport3D	976(33)	335 (34.32%)
	Test case 6	Matterport3D	996(31)	519 (52.11%)
	Test case 7	Matterport3D	1034(28)	930 (89.94%)

information for our test cases is shown in Table 4.1. The table shows the type of background model, the number of frames, the video length, and the number of frames which contain moving objects.

We only generate one test case on the virtual environment from Unity3D asset store as the model is a purely virtual environment. To simulate the real world better, we create six test cases with different scenarios from the Replica and Matterport3D datasets. The test cases are divided into two parts, fully dynamic and previously static, based on whether the object is moving when the camera first captures it. These two scenarios are common and representative in real life. The frame rate for each generated video is set to be no less than 30 fps. However, the actual speed for video generation depends on the total number of models to be rendered and the computational ability of the device. The frame rate in test case 1 is lower than others, as it is an outdoor virtual environment containing many models (trees and grass) to be rendered. All the objects that we have generated for the test cases shown in Table 4.1 are rigid.

Table 4.2: OVERVIEW OF SELECTED TUM TEST CASES

	Frame Number	Video length (s)
walking_static	2484	24.83
walking_xyz	2884	28.83
walking_rpy	3061	30.61
walking_halfsphere	3582	35.81
sitting_xyz	4251	42.50
sitting_rpy	2748	27.48

4.1.3 Selected public test cases

To show our performance on a standard benchmark, we select some test cases from the TUM RGB-D SLAM Dataset that contains human motions in the scene. Table 4.2 shows the overview of the selected test cases in TUM.

Different from our generated test cases, which focus on the problem of dynamic objects covering the view from the camera, the TUM dataset also suffers from other issues that may influence the estimation, including degenerate two-view geometry because of pure rotation, and the problem of textureless surfaces.

4.2 Motion of previously static objects

In this section, we perform tests with a previously static object starting to move while in the view of the camera. We speculate this situation to commonly occur, e.g., if a person or pet rests then gets up and moves as the camera comes closer. We have tested three different scenarios. In the first type of scenario, ORB-SLAM2 does not lose tracking, but accuracy is affected. In the second type of scenario, ORB-SLAM2 may lose tracking but

Figure 4.1: A sample image from Scenario 1.

can typically re-localize quickly based on previous views. In the last type of scenario, ORB-SLAM2 typically requires re-localization and loop closure. The objects in the synthetic test cases are rigid despite the fact that animals are articulated and deformable in reality.

In Scenario 1, a foreground object covers most of the view, and hence features on the object will be tracked. Our test case sequence contains 1974 frames. The test results of DOE-SLAM, DynaSLAM and ORB-SLAM2 for this scenario are shown in Figure 4.2 and Table 4.3. The left-top sub-graph shows the whole estimated trajectory for all three systems, as well as the ground truth. To show the results in detail, we zoom-in to the part of the trajectory that is influenced by the moving object (in the orange box) in the three corresponding sub-graphs for each SLAM system. The object is static at the beginning, and the estimations from ORB-SLAM2 and DOE-SLAM are almost the same. DynaSLAM loses tracking quickly due to the lack of background feature points. However, when the object starts to move, shown in the close up in Figure 4.2, the estimated trajectory by

ORB-SLAM2 drifts while DOE-SLAM keeps tracking the position of the camera and is nearly unaffected. We can see that if a moving object dominates the view, ORB-SLAM2 may not get lost, but the camera location is not reliable. In ORB-SLAM2, the features located on the background will be treated as outliers because the features on the foreground object occupy the most part of the frame. We calculated the RMSE of the object pose estimation (see Table 4.3). The accuracy of the object pose estimation is lower than of the camera pose estimation. In many of the frames, the object contains fewer features than the background, or the camera can not capture enough background features, and hence we predict the object motion with some reasonable error.

Table 4.3: COMPARISON OF THE RMSE OF ATE [CM] FOR CAMERA, OBJECT, AND THE NUMBER OF LOST FRAMES IN SCENARIO ONE.

	ATE RMSE	Object RMSE	lost frames
ORB-SLAM2	4.62	n/a	0
DynaSLAM	2.17	n/a	922 (46.7%)
DOE-SLAM	**1.67**	5.91	0

In Scenario 2 again, a foreground object covers most of the view, and hence features on the object will be tracked. However, this time the occlusion is severe enough such that without dynamic object handling, camera pose estimation will fail. We generated the test case from the Replica-Dataset in Unity3D to assess this scenario with 1596 frames. Figure 4.4 and Table 4.4 show the results that we have obtained for this scenario. The object is moving during the trajectory in the orange box in the top-left sub-graph. The result from ORB-SLAM2 shows that, when the object starts to move, the camera pose estimation starts to be affected. After several frames, tracking in ORB-SLAM2 fails as

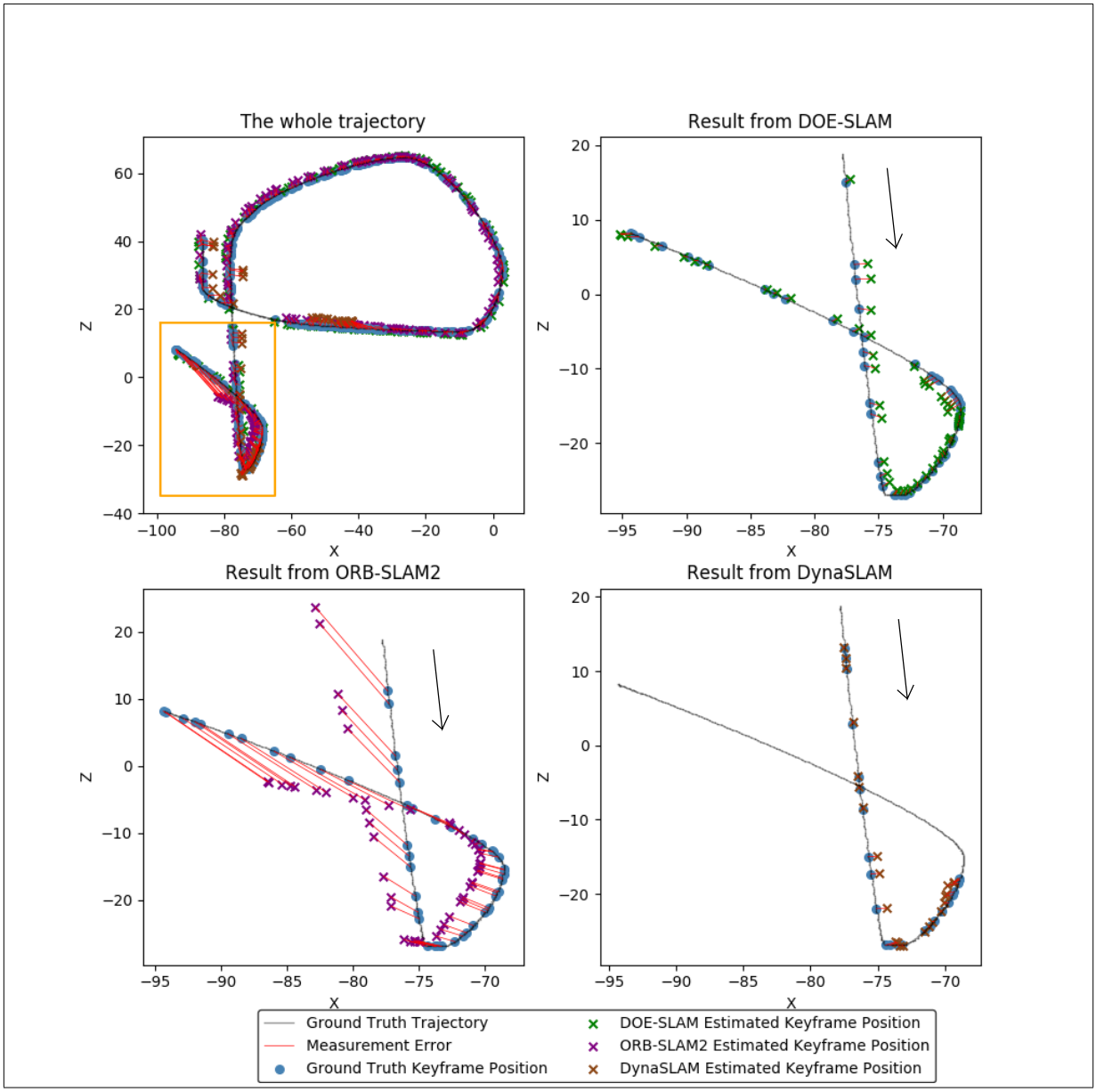


Figure 4.2: The test result for scenario 1. The arrow shows the direction of travel along the trajectory of the camera. The zoom in sub-graph on the top-right, bottom-left, and bottom-right show the section of the camera trajectory where the image frames contain the moving object.

Figure 4.3: A sample image from Scenario 2.

Table 4.4: COMPARISON OF THE RMSE OF ATE [CM] FOR CAMERA, OBJECT, AND THE NUMBER OF LOST FRAMES IN SCENARIO SCENARIO TWO.

	RMSE	Object RMSE	lost frames
ORB-SLAM2	23.34	n/a	39 (2.44%)
DynaSLAM	86.19	n/a	197 (12.34%)
DOE-SLAM	**18.05**	30.88	0

can be seen from the part of the trajectory curve without estimated blue camera positions. After the moving object passes by, the system re-localizes immediately as the camera has visited this location before. DynaSLAM gets lost once the object covers the camera. Our DOE-SLAM not only always keeps tracking, but also estimates the trajectory to good accuracy. Table 4.4 provides the RMSE and the number of lost frames for ORB-SLAM2, DynaSLAM, and DOE-SLAM. In this scenario, our method outperforms both comparison methods in terms of accuracy and robustness.

Scenario 3 contains a longer period where the moving object blocks the camera view

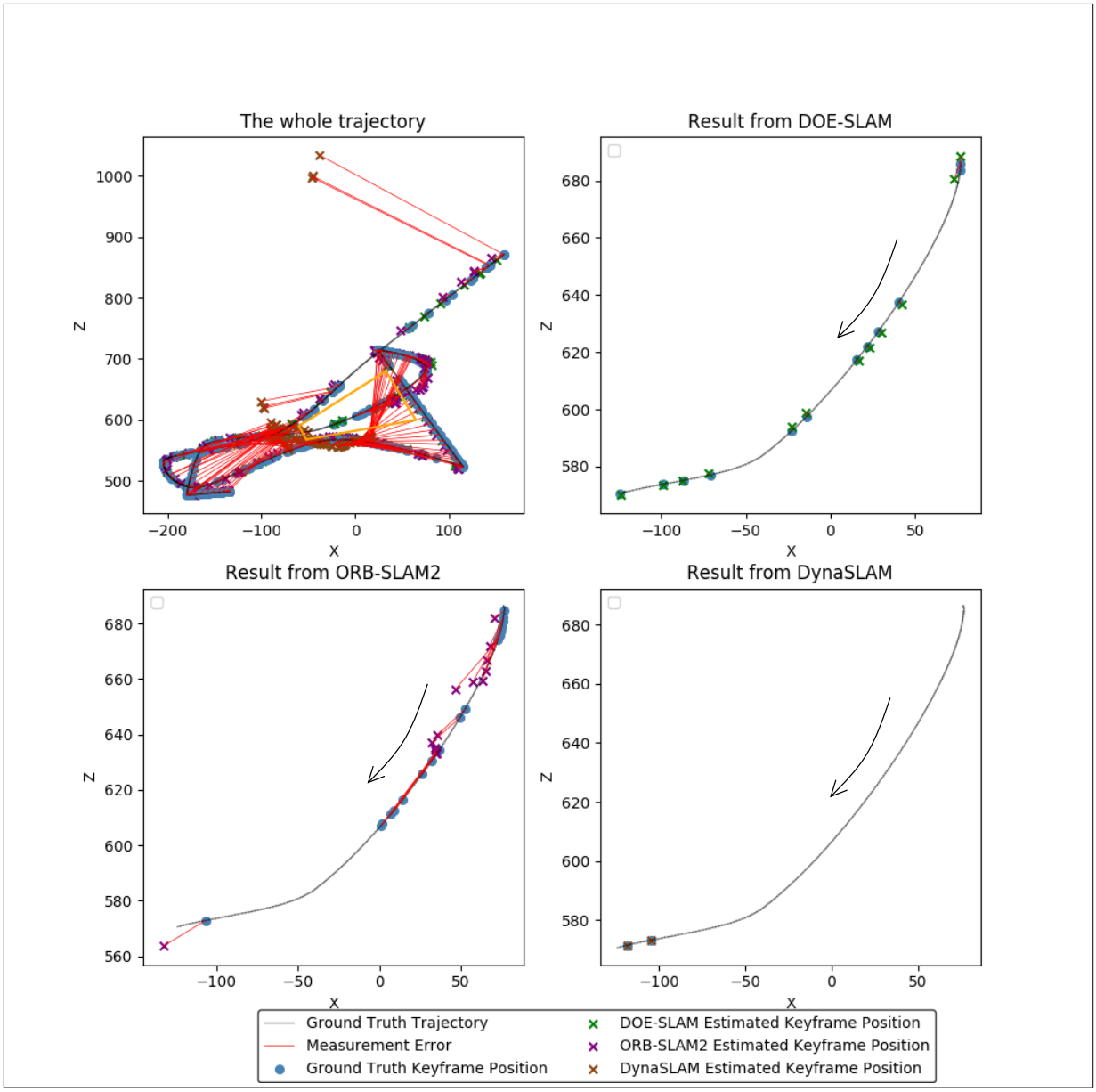


Figure 4.4: The test result for scenario 2. The arrow shows the direction of travel along the trajectory of the camera. The zoom in sub-graph on the top-right, bottom-left, and bottom-right show the section of the camera trajectory where the image frames contain the moving object.

Figure 4.5: A sample image from Scenario 3.

of the background. The scenario also provides the opportunity of loop closure after the moving object moves out of the view. Loop closing is the act of correctly asserting that a device has returned to a previously visited location. The test case is generated from the Matterport3D dataset with 2385 frames. The results are shown in Figure 4.6 and Table 4.5. We run the scenario multiple times and sometimes, ORB-SLAM2 tracks from the object without getting lost, but the result is inaccurate. However, most of the time, ORB-SLAM2 directly stops tracking until a loop closure occurs (performs the same as DynaSLAM). DynaSLAM gets lost in the area with the moving object marked by the orange box. In contrast, our DOE-SLAM works well in this scenario. Despite the scale drift in frames after the object passes by, loop closure eventually corrects and minimizes this scale error. Table 4.5 shows that our DOE-SLAM performs significantly better than ORB-SLAM2. The table also shows that the RMSE for ORB-SLAM2 is lower when tracking

Table 4.5: COMPARISON OF THE RMSE OF ATE [CM] FOR CAMERA, OBJECT, AND THE NUMBER OF LOST FRAMES IN SCENARIO SCENARIO THREE.

	RMSE	Object RMSE	lost frames
ORB-SLAM2	176.46	n/a	0
ORB-SLAM2 (lost)	60.97	n/a	554 (23.23%)
DynaSLAM	**39.44**	n/a	639 (26.79%)
DOE-SLAM	48.66	53.40	0

is lost than if the method keeps tracking, and DynaSLAM gets a better result than ours. This is due to the fact that we calculate the RMSE in matched time steps only, but if the system is lost when a significant error occurs, this will lead to a reduced overall error. We have to consider that a monocular camera can not capture the depth of the environment, and a scale for all the features points is set during initialization. After the object passes through the view, the system may create new background points but with a different scale factor, and hence the scale drifts even further.

In three test cases, the accuracy of DynaSLAM is higher than ORB-SLAM2 in scenario 1 and 3, and even close to our DOE-SLAM in scenario 3. However, DynaSLAM is far less stable than ours in these test cases. While the foreground moving object obstructs the camera, tracking is easily lost. However, our system performs better than both DynaSLAM and ORB-SLAM2 as it keeps tracking successfully in these scenarios.

We also compared the computational time required by DOE-SLAM compared to ORB-SLAM2, and DynaSLAM. Table 4.6 shows the average operation time for each frame without the object segmentation. The test result shows that Doe-SLAM is slower than ORB-SLAM2 but faster than DynaSLAM.

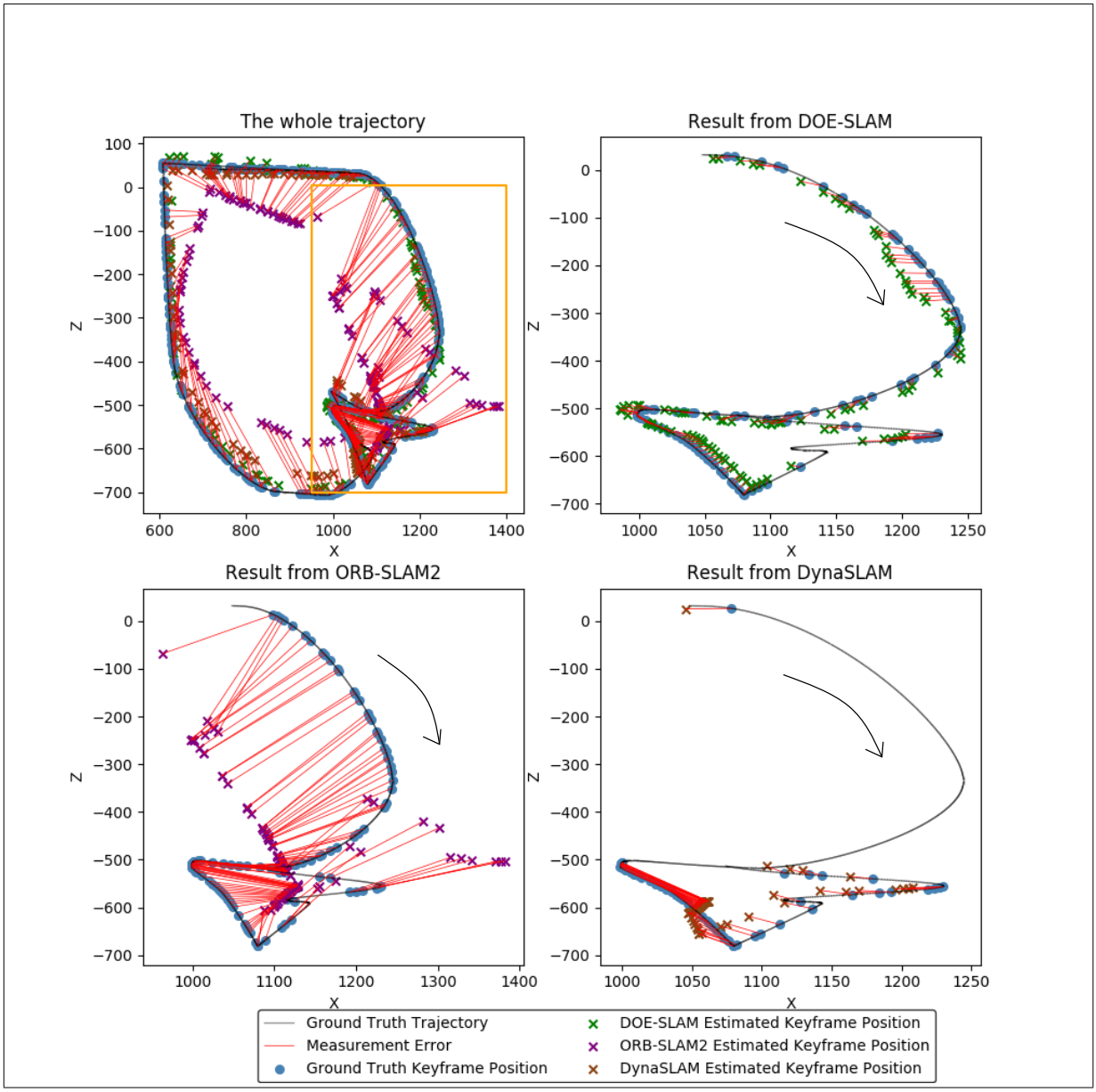


Figure 4.6: The test result for scenario 3. The arrow shows the direction of travel along the trajectory of the camera. The zoom in sub-graph on the top-right, bottom-left, and bottom-right show the section of the camera trajectory where the image frames contain the moving object.

Table 4.6: COMPARISON OF COMPUTATION TIME PER FRAME [MS]

	Scenario 1	Scenario 2	Scenario 3
ORB-SLAM2	20.5	25.06	20.65
DynaSLAM	31.12	33.56	27.34
DOE-SLAM	**25.21**	**29.8**	**26.13**

4.3 Fully dynamic objects

We created scenarios where a moving object appears in the view and the object is always in motion when seen by the camera. As the moving object becomes dominant in the view, the scale estimation for a mono camera is impacted, which may lead to scale drift between the map before the dynamic object comes into view and when the object leaves the view. Loop closure can eliminate scale drift, and hence we generated four test cases, three without and one with a loop. Also the objects in the synthetic test cases are rigid despite the fact that animals are articulated and deformable in reality. The objects accelerate at the beginning and keep constant velocity until they pass the camera.

We created three test cases (test case 4, 6, and 7 in Section 4.1.2) without a loop from the Replica-Dataset and Matterport3D, and we discuss one of them (test case 4) in detail (see Figure 4.7). We can see from the graph that ORB-SLAM2 is heavily influenced by the moving object, as the moving features affect the image alignment. DynaSLAM not only gets lost for some frames but also tracks at a low accuracy, as the captured background features are too few to be tracked. Our DOE-SLAM produces a high accuracy trajectory and is stable in tracking without getting lost.

We generated another test case but this time with a loop from Matterport3D with 976

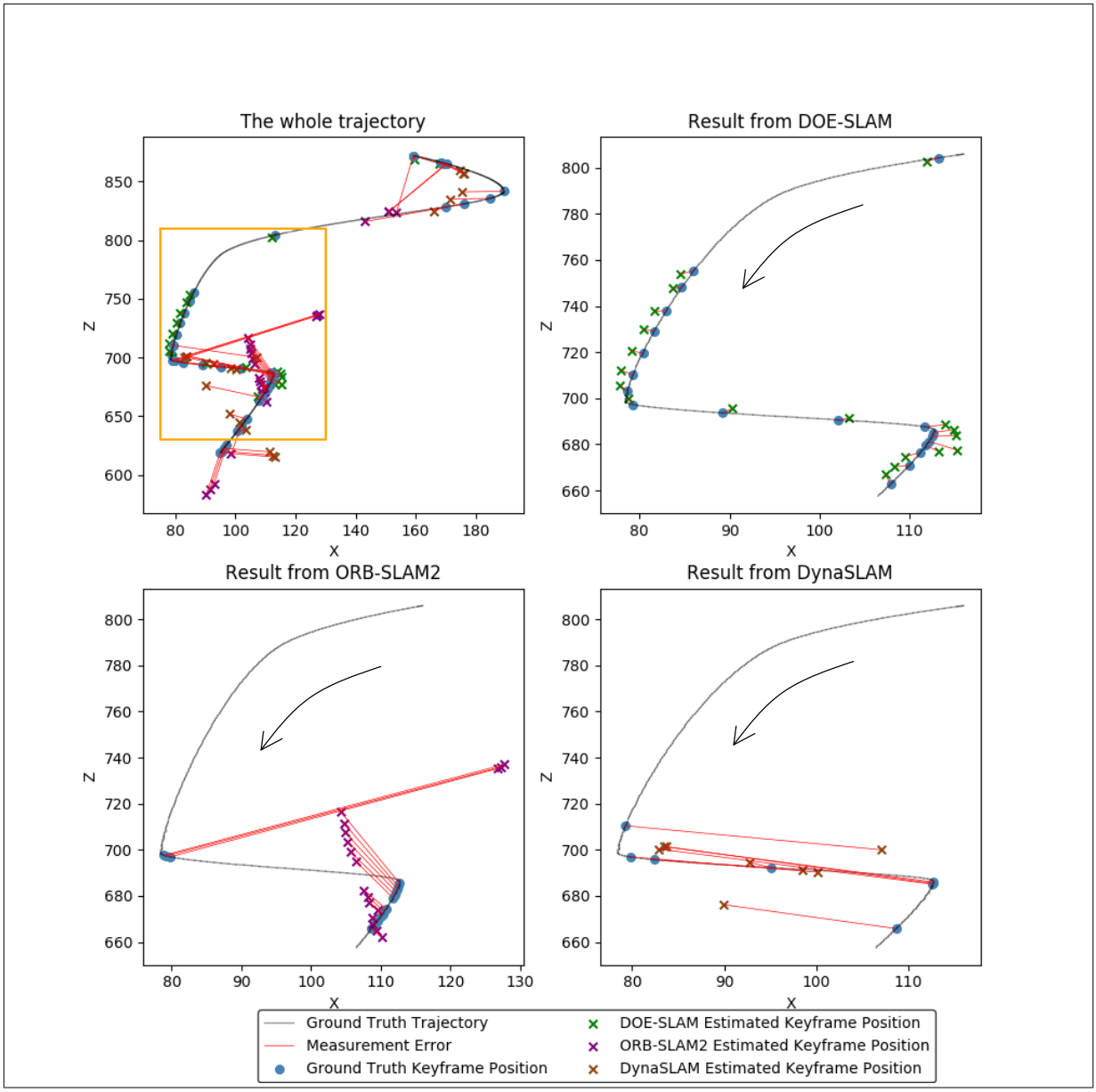


Figure 4.7: The test result for scenario 4. The arrow shows the direction of travel along the trajectory of the camera. The zoom in sub-graph on the top-right, bottom-left, and bottom-right show the section of the camera trajectory where the image frames contain the moving object.

Table 4.7: COMPARISON OF THE RMSE OF ATE [CM] AND THE NUMBER OF LOST FRAMES FOR FULLY DYNAMIC OBJECT CASES

		Test case 4	Test case 5	Test case 6	Test case 7
ORB-SLAM2	RMSE	86.98	31.30	6.79	100.72
	lost frames	0	0	0	0
DynaSLAM	RMSE	16.40	25.76	9.29	**21.29**
	lost frames	31	39	695	612
DOE-SLAM	RMSE	**16.11**	**8.61**	**5.17**	66.29
	lost frames	0	0	0	0

frames in total (test case 5 in Table 4.1). Figure 4.8 shows that ORB-SLAM2 produces large errors over the complete trajectory. Although the result from DynaSLAM is quite accurate, DynaSLAM loses tracking during the period when the object obstructs the camera.

We run each test case on each system five times, and the average RMSE of ATE (see Table 4.7) shows that our DOE-SLAM outperforms the two competitors based on accuracy. We also observed the average number of frames where tracking is lost. In these four test cases, our DOE-SLAM has the highest accuracy with the lowest RMSE and high robustness as it does not lose tracking in any frame. Although ORB-SLAM2 does not get lost, it treats the object as always static and uses object features for tracking, which seriously affects the accuracy. On the contrary, DynaSLAM outperforms ORB-SLAM2 and our DOE-SLAM in accuracy on test case 7. However, it is unstable because in 612 frames tracking is lost. The reason for its instability is that it excludes object features from the camera pose estimation.

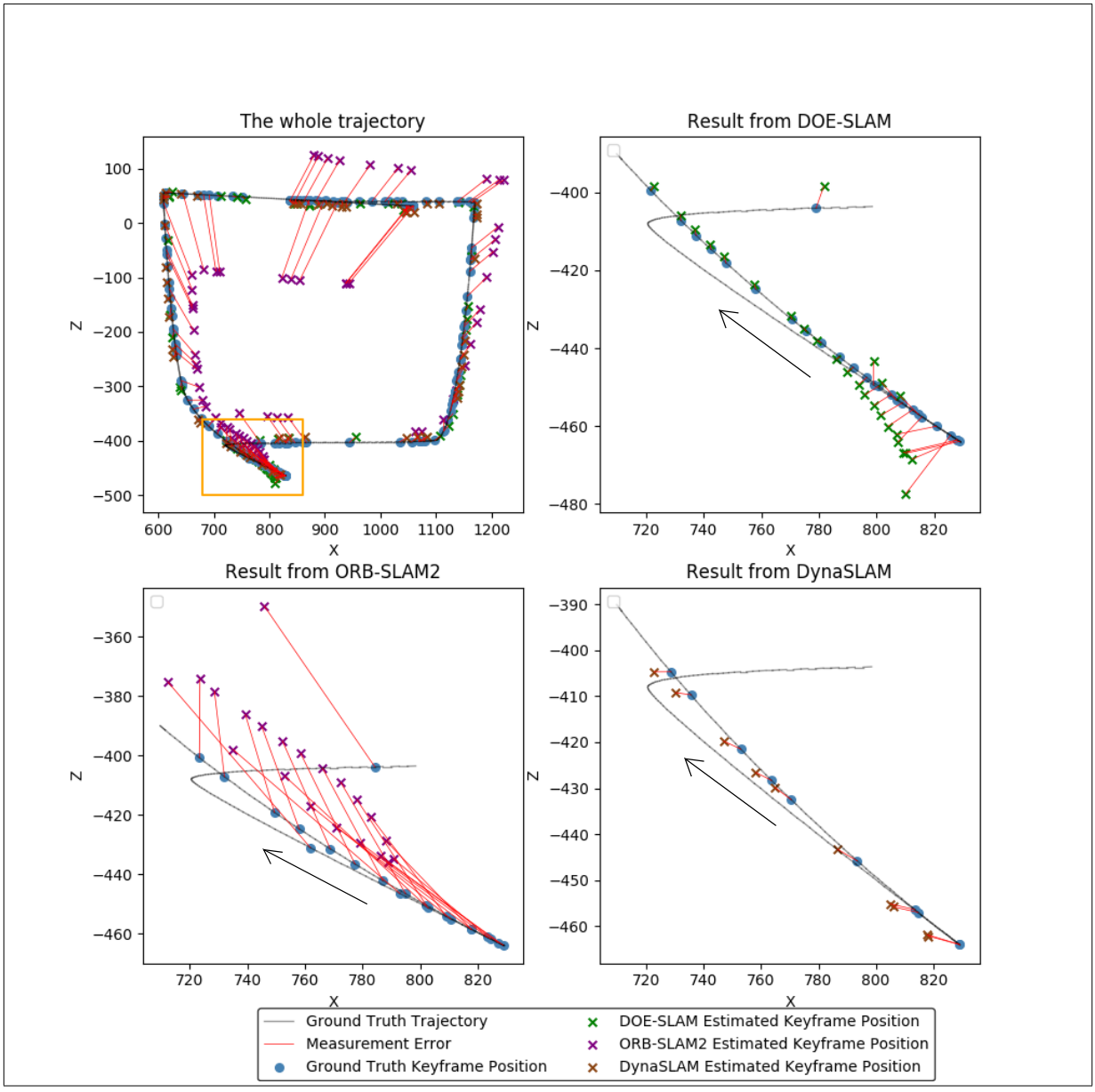


Figure 4.8: The test result for scenario 5. The arrow shows the direction of travel along the trajectory of the camera. The zoom in sub-graph on the top-right, bottom-left, and bottom-right show the section of the camera trajectory where the image frames contain the moving object.

Table 4.8: COMPARISON OF THE RMSE OF ATE [CM] IN SELECTED TUM DATASETS

		ORB-SLAM2	DynaSLAM	DOE-SLAM
walking_static	RMSE AVG	1.74	**0.49**	0.58
	RMSE VAR	0.208	0.010	0.005
walking_xyz	RMSE AVG	1.41	1.53	**1.05**
	RMSE VAR	0.031	0.030	0.031
walking_rpy	RMSE AVG	6.38	**4.81**	5.71
	RMSE VAR	1.579	1.263	2.476
walking_halfsphere	RMSE AVG	1.79	1.77	**1.65**
	RMSE VAR	0.016	0.022	0.009
sitting_rpy	RMSE AVG	2.40	2.02	**1.81**
	RMSE VAR	0.276	0.020	0.320
sitting_xyz	RMSE AVG	0.99	1.17	**0.62**
	RMSE VAR	0.014	0.007	0.006

4.4 TUM dataset

We select some appropriate TUM datasets (Section 4.1.3) as test cases to evaluate DOE-SLAM in common scenarios.

Table 4.8 shows the test results on selected TUM datasets. For each test case, we test five times on each system. The average and variance of RMSE are calculated. From the table, we can see that our DOE-SLAM outperforms the comparison methods in most of the test cases (*walking_xyz*, *walking_halfsphere*, *sitting_rpy*, and *sitting_xyz*) with a lower average RMSE. However, in some cases, DynaSLAM performs better. In *walking_static* test case, the camera is static, and there are plenty of background features that can be captured. DynaSLAM gets the lowest average RMSE in this test case as DynaSLAM

dilates masks to cover more features near the edge area. We can also see from the result that although humans are not rigid, by treating the human as a rigid body our system can improve its performance, especially if the obstruction of the camera view last only a few frames.

However, in some test cases, the moving object is not the only issue or even not the main issue that weakens the estimation. In *walking_xyz*, *walking_halfsphere* and *sitting_xyz*, the camera pose measurement also suffers from the problem of textureless surfaces, because the camera only captures the white wall or the floor in several frames, which causes the system to lose tracking. DynaSLAM performs even worse than ORB-SLAM2 in these scenarios. It masks out the object feature points but the background scene is textureless, which leads to not enough features for tracking. In *walking_rpy* and *sitting_rpy*, the camera is rotated along the principal axes (roll-pitch-yaw) at the same position [75]. Pure rotation is another issue that limits the accuracy of estimation in these scenarios in addition to moving objects. The result for these two test cases shows that DOE-SLAM performs the same as DynaSLAM or even worse, and the variance of RMSE is the highest of three systems, which means the estimation is unstable. As we predict the camera pose from object motion, in the high-uncertain scenario (like in pure rotation, the two-view geometry is invalid), our system contains more uncertainty, including object pose estimation, and camera pose prediction. As mentioned in Chapter 3, the scale difference between the object and background also affects the accuracy of our estimation strategy.

The number of lost frames is counted in Table 4.9. The more lost frames the system suffers, the less robust the system is. The test result shows that DOE-SLAM outperforms

Table 4.9: COMPARISON OF THE NUMBER OF LOST FRAMES IN SELECTED TUM DATASETS (ONLY SHOWS THE CASES THAT CAMERA GETS LOST)

	ORB-SLAM2	DynaSLAM	DOE-SLAM
walking_static	44	42	2
walking_rpy	151	82	4

the other two systems in robustness. We only show the cases for which tracking is lost for some frames. For *walking_xyz*, and *sitting_xyz*, all three systems work well without lost frames. For *sitting_rpy*, and *walking_halfsphere*, the camera is hard to be initialized based on the reason mentioned before. So all three systems start tracking after the most challenging part passes. The result is not really relevant here.

4.5 Summary

In this chapter, we first introduce the test cases that we have selected for evaluation. We generate 7 new test cases from the Unity3D asset store [69], the Replica dataset [73], and the Matterport3D [13] dataset. We also select 6 test cases from the public TUM dataset [75]. Then we present the experiments that compare our system with two state-of-the-art monocular vSLAM systems: ORB-SLAM2, and DynaSLAM. Some new generated test cases and selected cases of a public data set are employed to measure the performance.

In our experiments we show that, in dynamic environments, our DOE-SLAM performs better than the state-of-the-art SLAM systems in accuracy and robustness. DOE-SLAM achieves the highest accuracy in most of test cases and it loses tracking in the minimal number of frames. The experiment also shows that our novel strategy using object motion

to predict camera pose when a moving object covers the camera, is feasible. However, our system has limitations during pure rotation which still needs to be solved in future work.

In the next chapter, we present the conclusions of this thesis and our system. We then analyze the pros and cons of our DOE-SLAM and state the improvement in dynamic environments of DOE-SLAM. Some further research directions are provided in future works.

Chapter 5

Conclusion

In this chapter, we summarize our contributions in this thesis in Section 5.1. We then discuss the pros and cons of our system according to our experiments. Some future research directions are given in Section 5.3.

5.1 Summary

In this thesis, we investigate the development of SLAM within dynamic environments. SLAM is a popular strategy to deal with localization and mapping problems in the field of computer vision and robotics. Augmented reality, robotics route planning, and many robotic navigation systems adopt SLAM as their main strategy to self-localize. To improve the accuracy of SLAM systems, many types of sensors are utilized, including but not limited to a monocular camera, a stereo camera, a depth camera, and an IMU. However, there are some bottlenecks in SLAM which limit the development. Monocular based vSLAM suffers from degeneracies in pure rotation, tracking problems due to textureless surfaces and dynamic environments, as well as inefficiencies during initialization. These problems

affect the estimation accuracy and the robustness of the system. Up to now, the common strategy to handle the dynamic environment problem is masking out all the moving objects. By only focusing on background features, the accuracy is improved, but the system also loses robustness.

We present a novel SLAM system in this thesis to overcome many of the problems due to dynamic environments even if the moving object obstructs the camera for a short period of time. Our system first segments the object in the image frame. During the tracking period, the system first tries to track the background features. If this succeeds, we also estimate the object motion in the frame. The background and the object are tracked separately to avoid the influence of moving objects. If the object is moving, we optimize the object motion and localize the camera pose to only rely on the background features. When the moving object is close enough to obstruct the camera so that the system can not capture enough background for tracking, our system recovers the camera pose from the object motion. If the camera moves into a place previously visited, the system optimize the scale drift by using Bundle Adjustment.

To evaluate the performance of our system, we utilize Unity3D to generate several test cases from the Replica dataset, and the Matterport3D dataset, which fully expose the dynamic environment problem and provide ground truth as we require. We employ some test cases from the commonly-used public TUM dataset which contain human motions.

ORB-SLAM2 and DynaSLAM are selected for comparison to show our improvement as they are two state-of-the-art monocular vSLAM systems, and DynaSLAM also focuses on the dynamic environment problem. The experiments contain 13 test cases including 6

test cases from the TUM dataset. For the test cases we made, we designed two different scenarios: one is with the object moving during the whole video, and the other is with the object starting to move as it is observed by the camera. The experiments show that our DOE-SLAM achieves the highest accuracy in most of the test cases, and is the most robust of three SLAM systems. The results of the computation time show that our DOE-SLAM is slower than ORB-SLAM2, but faster than DynaSLAM. DOE-SLAM can still run in real time.

5.2 Contributions

We have the following contributions in my research work:

- We present a new strategy to estimate the camera pose and object motion simultaneously in SLAM system.
- We put forward a new method to deal with the dynamic environment problem by using the moving object to improve the robustness and accuracy of the system. And we implemented a new SLAM system on top of the ORB-SLAM2 to adapt to dynamic environment.
- For a better evaluation to fully exploit our improvements, we also generated several datasets for testing.

From the experiment, our DOE-SLAM system outperforms the state-of-the-art ORB-SLAM2 and DynaSLAM in robustness and accuracy in dynamic environment.

5.3 Limitation and future works

DOE-SLAM improves on the state-of-the-art systems in our comparison by increasing the accuracy and reducing the number of lost frames. However, our system still has some limitations, which can be seen in the experiments. Pure rotation is a common issue for monocular vSLAM systems. Monocular vSLAM employs two-view geometry to reconstruct 3D points, but pure rotation leads to a degeneracy in essential matrix estimation for two-view geometry. As a feature-based vSLAM system, DOE-SLAM relies on image features to align images. If the camera captures a textureless image, the system can not extract enough features for tracking. Except for the above issues, which are the common problems for feature-based monocular vSLAM, we provide some future research directions.

Multi-object tracking is a notable direction for future works. Our system only tracks the main object in the view. The system needs a number of feature points to estimate the pose accurately. We only consider the dominant object which is the largest object in the view to make sure the system is able to capture enough feature points. However, relying on only one object is sometimes unstable. If the object suddenly changes the motion or the object is textureless in one or some of the surfaces, the accuracy of camera pose prediction can be affected. By multi-object tracking, the system can rely on more than one object to recover the camera pose which would make the system more stable.

Motion simulation is a widely researched topic. Many different models are published to simulate object motions. In DOE-SLAM, we utilize a constant motion model to simulate and predict the object motion. It works well in our experiments as the camera is obstructed by the object for a short period. To make the system more general by being able to

follow the moving object for a long period, a well designed motion simulation system is necessary, as our strategy to predict the camera pose is dependent on the prediction of the object motion. An accurate object motion simulation is able to improve the camera pose prediction.

Scale variation is a drawback of all monocular vSLAM systems. The scale uncertainty of the monocular camera means a monocular vSLAM system can not estimate the real scale of the scene. Our DOE-SLAM builds the object and background models separately. There usually is a scale difference between two models, which is known to affect the accuracy of camera pose prediction. Although the real scale is unknown, one could design a strategy to minimize the scale difference between the object and the background. We believe that minimizing the scale difference can increase the accuracy of the pose estimation.

Non-rigid body Non-rigid object tracking is a current research topic in computer vision. Our system currently treats each object as rigid body for tracking regardless if the object undergoes deformation. In some cases like human tracking, our system will lose tracking if the person deforms such the features can no longer be matched. If non-rigid body tracking can be embedded into our system, it may improve the ability to track objects and our system may perform better in more general scenarios.

References

[1] Iro Armeni, Alexander Sax, Amir Roshan Zamir, and Silvo Savarese. Joint 2D-3D-Semantic data for indoor scene understanding. *ArXiv*.

[2] Ronald T Azuma. A survey of Augmented Reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision*, pages 404–417. Springer, 2006.

[5] Berta Bescos, José M Fácil, Javier Civera, and José Neira. DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083, 2018.

[6] Mark Billinghurst, Adrian Clark, Gun Lee, et al. A survey of Augmented Reality. *Foundations and Trends® in Human–Computer Interaction*, 8(2-3):73–272, 2015.

[7] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison. CodeSLAM—learning a compact, optimisable representation for dense visual SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2560–2568, 2018.

[8] Nikolas Brasch, Aljaz Bozic, Joe Lallemand, and Federico Tombari. Semantic Monocular SLAM for Highly Dynamic Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 393–400. IEEE, 2018.

[9] Kenneth M Brown. A Quadratically Convergent Newton-like method Based Upon Gaussian Elimination. *SIAM Journal on Numerical Analysis*, 6(4):560–569, 1969.

[10] Natalie Bursztyn, A Walker, B Shelton, and Joel Pederson. Increasing Undergraduate Interest to Learn Geoscience with GPS-based Augmented Reality Field Trips on Students' Own Smartphones. *Geological Society of America Today*, 27(5):4–11, 2017.

[11] Su Cai, Feng-Kuang Chiang, Yuchen Sun, Chenglong Lin, and Joey J Lee. Applications of Augmented Reality-based Natural Interactive Learning in Magnetic Field Instruction. *Interactive Learning Environments*, 25(6):778–791, 2017.

[12] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European Conference on Computer Vision*, pages 778–792. Springer, 2010.

[13] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D Data in Indoor Environments. *3D Vision*, 2017.

[14] Alexander M Clark and Matthew TG Clark. Pokemon Go and Research: Qualitative, Mixed Methods Research, and the Supercomplexity of Interventions, 2016.

[15] Andrew I Comport, Éric Marchand, and François Chaumette. A real-time tracker for markerless Augmented Reality. In *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR), 2003. Proceedings.*, pages 36–45. IEEE, 2003.

[16] Joseph DeGol, Timothy Bretl, and Derek Hoiem. Improved structure from motion using fiducial marker matching. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 273–288, 2018.

[17] Petar M Djuric, Jayesh H Kotecha, Jianqui Zhang, Yufei Huang, Tadesse Ghirmai, Mónica F Bugallo, and Joaquin Miguez. Particle Filtering. *IEEE Signal Processing Magazine*, 20(5):19–38, 2003.

[18] Garry A Einicke and Langford B White. Robust Extended Kalman Filtering. *IEEE Transactions on Signal Processing*, 47(9):2596–2599, 1999.

[19] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, 2017.

[20] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 834–849. Springer, 2014.

[21] Michele Fiorentino, Raffaele de Amicis, Giuseppe Monno, and Andre Stork. Spacedesign: A Mixed Reality workspace for aesthetic industrial design. In *Pro-*

ceedings. International Symposium on Mixed and Augmented Reality, pages 86–318. IEEE, 2002.

[22] Martin A Fischler and Robert C Bolles. Random Sample Consensus: A Paradigm for Model Fitting With Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[23] Wolfgang Friedrich, D Jahn, and L Schmidt. ARVIKA-Augmented Reality for Development, Production and Service. In *Proceedings. International Symposium on Mixed and Augmented Reality*, volume 2, pages 3–4. Citeseer, 2002.

[24] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[25] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017.

[26] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[27] Georg Gerstweiler, Emanuel Vonach, and Hannes Kaufmann. HyMoTrack: A mobile AR navigation system for complex indoor environments. *Sensors*, 16(1):17, 2016.

[28] 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 (CVPR)*, 2014.

[29] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 270–279, 2017.

[30] Michael Grupp. evo: Python package for the evaluation of odometry and SLAM. https://github.com/MichaelGrupp/evo. Accessed September, 2019.

[31] Ryo Hachiuma, Christian Pirchheim, Dieter Schmalstieg, and Hideo Saito. DetectFusion: Detecting and Segmenting Both Known and Unknown Dynamic Objects in Real-time SLAM. 2019.

[32] Felix G Hamza-Lup, Jannick P Rolland, and Charles Hughes. A distributed Augmented Reality system for medical training and simulation. *Energy, Simulation-Training, Ocean Engineering and Instrumentation: Research Papers of the Link Foundation Fellows, Vol. 4, Rochester Press.*, 2018.

[33] Richard I Hartley. In defense of the eight-point algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(6):580–593, 1997.

[34] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969, 2017.

[35] Aharon Bar Hillel, Ronen Lerner, Dan Levi, and Guy Raz. Recent progress in road and lane detection: a survey. *Machine Vision and Applications*, 25(3):727–745, 2014.

[36] Tobias Höllerer and Steve Feiner. Mobile Augmented Reality. *Telegeoinformatics: Location-based computing and services*, 21, 2004.

[37] Simon J Julier and Jeffrey K Uhlmann. New extension of the Kalman filter to nonlinear systems. In *Signal Processing, Sensor Fusion, and Target Recognition VI*, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[38] Andrej Karpathy, Stephen Miller, and Li Fei-Fei. Object discovery in 3d scenes via shape analysis. In *2013 IEEE International Conference on Robotics and Automation*, pages 2088–2095. IEEE, 2013.

[39] Sung Lae Kim, Hae Jung Suk, Jeong Hwa Kang, Jun Mo Jung, Teemu H Laine, and Joonas Westlin. Using Unity 3D to Facilitate Mobile Augmented Reality Game Development. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 21–26. IEEE, 2014.

[40] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings. International Symposium on Mixed and Augmented Reality*, pages 225–234. IEEE, 2007.

[41] Christian Koch, Matthias Neges, Markus König, and Michael Abramovici. Natural Markers for Augmented Reality-based Indoor Navigation and Facility Maintenance. *Automation in Construction*, 48:18–30, 2014.

[42] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g 2 o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.

[43] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[44] Shiqi Li, Chi Xu, and Ming Xie. A robust O (n) solution to the perspective-n-point problem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1444–1450, 2012.

[45] Haomin Liu, Guofeng Zhang, and Hujun Bao. Robust keyframe-based monocular SLAM for Augmented Reality. In *Proceedings. International Symposium on Mixed and Augmented Reality*, pages 1–10. IEEE, 2016.

[46] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.

[47] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. Pose estimation for Augmented Reality: a hands-on survey. *IEEE Transactions on Visualization and Computer Graphics*, 22(12):2633–2651, 2015.

[48] Jorge J Moré. The Levenberg-Marquardt Algorithm: Implementation and Theory. In *Numerical Analysis*, pages 105–116. Springer, 1978.

[49] Rafael Muñoz-Salinas, Manuel J Marín-Jimenez, Enrique Yeguas-Bolivar, and Rafael Medina-Carnicer. Mapping and localization from planar markers. *Pattern Recognition*, 73:158–171, 2018.

[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.

[51] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: An open-source slam system for monocular, stereo, and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[52] Nassir Navab, Tobias Blum, Lejing Wang, Asli Okur, and Thomas Wendler. First deployments of Augmented Reality in operating rooms. *Computer*, 45(7):48–55, 2012.

[53] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. DTAM: Dense tracking and mapping in real-time. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2320–2327. IEEE, 2011.

[54] Pauline C Ng and Steven Henikoff. SIFT: Predicting amino acid changes that affect protein function. *Nucleic Acids Research*, 31(13):3812–3814, 2003.

[55] Lachlan Nicholson, Michael Milford, and Niko Sünderhauf. QuadricSLAM: Dual quadrics from object detections as landmarks in object-oriented SLAM. *IEEE Robotics and Automation Letters*, 4(1):1–8, 2018.

[56] Jason D O'Grady. *Apple Inc.* ABC-CLIO, 2009.

[57] Onur Özyeşil, Vladislav Voroninski, Ronen Basri, and Amit Singer. A survey of structure from motion. *Acta Numerica*, 26:305–364, 2017.

[58] Bernd Pfrommer and Kostas Daniilidis. TagSLAM: Robust SLAM with fiducial markers. *arXiv preprint arXiv:1910.00679*, 2019.

[59] Kai Qiu, Yunfeng Ai, Bin Tian, Bin Wang, and Dongpu Cao. Siamese-ResNet: Implementing Loop Closure Detection based on Siamese Network. In *2018 IEEE Intelligent Vehicles Symposium*, pages 716–721. IEEE, 2018.

[60] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[61] 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.

[62] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-assisted Intervention*, pages 234–241. Springer, 2015.

[63] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443. Springer, 2006.

[64] Martin Rünz and Lourdes Agapito. Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4471–4478. IEEE, 2017.

[65] Martin Runz, Maud Buffier, and Lourdes Agapito. Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects. In *Proceedings. International Symposium on Mixed and Augmented Reality*, pages 10–20. IEEE, 2018.

[66] Muhamad Risqi U Saputra, Andrew Markham, and Niki Trigoni. Visual SLAM and structure from motion in dynamic environments: A survey. *ACM Computing Surveys (CSUR)*, 51(2):1–36, 2018.

[67] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for e AI research. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9339–9347, 2019.

[68] Raluca Scona, Mariano Jaimez, Yvan R Petillot, Maurice Fallon, and Daniel Cremers. Staticfusion: Background reconstruction for dense rgb-d slam in dynamic environments. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[69] Shapes. Nature Starter Kit 2. https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977#releases. Accessed September, 2019.

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

[71] P Sivakumar and S Meenakshi. A review on image segmentation techniques. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 5(3):641–647, 2016.

[72] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating Uncertain Spatial Relationships in Robotics. In *Autonomous Robot Vehicles*, pages 167–193. Springer, 1990.

[73] Julian Straub and et al. The Replica Dataset: A Digital Replica of Indoor Spaces. *arXiv preprint arXiv:1906.05797*, 2019.

[74] Michael Strecke and Jorg Stuckler. EM-Fusion: Dynamic Object-Level SLAM with Probabilistic Data Association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5865–5874, 2019.

[75] Jürgen Sturm, Wolfram Burgard, and Daniel Cremers. Evaluating Egomotion and Structure-from-Motion Approaches Using the TUM RGB-D Benchmark. In *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2012.

[76] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 573–580. IEEE, 2012.

[77] Yuxiang Sun, Ming Liu, and Max Q-H Meng. Motion removal for reliable RGB-D SLAM in dynamic environments. *Robotics and Autonomous Systems*, 108:115–128, 2018.

[78] Jeffrey S Swayze, Joshua Young, Geoffrey S Strobl, and Andrew Beckman. Surgical System with Augmented Reality Display, June 21 2018. US Patent App. 15/383,004.

[79] Keijiro Takahashi. Pcx-Point Cloud Importer/Renderer for Unity. https://github.com/keijiro/Pcx. Accessed September, 2019.

[80] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual SLAM algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, 2017.

[81] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6243–6252, 2017.

[82] Keisuke Tateno, Federico Tombari, and Nassir Navab. Real-time and scalable incremental segmentation on dense SLAM. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4465–4472. IEEE, 2015.

[83] PC Thomas and WM David. Augmented Reality: An application of heads-up display technology to manual manufacturing processes. In *Hawaii International Conference on System Sciences*, pages 659–669, 1992.

[84] Sebastian Thrun, Michael Montemerlo, Daphne Koller, Ben Wegbreit, Juan Nieto, and Eduardo Nebot. Fastslam: An Efficient Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.

[85] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International Workshop on Vision Algorithms*, pages 298–372. Springer, 1999.

[86] Vladyslav Usenko, Jakob Engel, Jörg Stückler, and Daniel Cremers. Direct visual-inertial odometry with stereo cameras. In *IEEE International Conference on Robotics and Automation (ICCV),*.

[87] DWF Van Krevelen and Ronald Poelman. A survey of Augmented Reality technologies, applications and limitations. *International Journal of Virtual Reality*, 9(2):1–20, 2010.

[88] Jonathan Ventura, Clemens Arth, Gerhard Reitmayr, and Dieter Schmalstieg. Global localization from monocular SLAM on a mobile phone. *IEEE Transactions on Visualization and Computer Graphics*, 20(4):531–539, 2014.

[89] Vassilios Vlahakis, M Ioannidis, John Karigiannis, Manolis Tsotros, Michael Gounaris, Didier Stricker, Tim Gleue, Patrick Daehne, and Luís Almeida. Archeoguide: an Augmented Reality guide for archaeological sites. *IEEE Computer Graphics and Applications*, 22(5):52–60, 2002.

[90] John Wang and Edwin Olson. AprilTag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4193–4198. IEEE, 2016.

[91] Sa Wang, Zhengli Mao, Changhai Zeng, Huili Gong, Shanshan Li, and Beibei Chen. A New Mmethod of Virtual Reality Based on Unity3D. In *2010 18th Iinternational Conference on Geoinformatics*, pages 1–5. IEEE, 2010.

[92] Zemin Wang, Qian Zhang, Jiansheng Li, Shuming Zhang, and Jingbin Liu. A computationally efficient semantic slam solution for dynamic scenes. *Remote Sensing*, 11(11):1363, 2019.

[93] Sabine Webel, Ulrich Bockholt, and Jens Keil. Design criteria for AR-based training of maintenance and assembly tasks. In *International Conference on Virtual and Mixed Reality*, pages 123–132. Springer, 2011.

[94] Giles Westerfield, Antonija Mitrovic, and Mark Billinghurst. Intelligent Augmented Rreality Training for Motherboard Assembly. *International Journal of Artificial Intelligence in Education*, 25(1):157–172, 2015.

[95] LH Wong et al. Mobile campus touring system based on AR and GPS: A case study of campus cultural activity. In *Proceedings of the 21st International Conference on Computers in Education*, 2013.

[96] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world Perception for Embodied Agents. In *Proceedings*

of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, 2018.

[97] Binbin Xu, Wenbin Li, Dimos Tzoumanikas, Michael Bloesch, Andrew Davison, and Stefan Leutenegger. Mid-fusion: Octree-based object-level multi-instance dynamic SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5231–5237. IEEE, 2019.

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

[99] Chao Yu, Zuxin Liu, Xin-Jun Liu, Fugui Xie, Yi Yang, Qi Wei, and Qiao Fei. DS-SLAM: A semantic visual SLAM towards dynamic environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1168–1174. IEEE, 2018.

[100] Fangwei Zhong, Sheng Wang, Ziqi Zhang, and Yizhou Wang. Detect-SLAM: Making object detection and slam mutually beneficial. In *IEEE Winter Conference on Applications of Computer Vision*, pages 1001–1010. IEEE, 2018.

[101] Huizhong Zhou, Danping Zou, Ling Pei, Rendong Ying, Peilin Liu, and Wenxian Yu. StructSLAM: Visual SLAM with building structure lines. *IEEE Transactions on Vehicular Technology*, 64(4):1364–1375, 2015.