Real-time Visual Simultaneous Localization and Mapping under Dynamic Environments

(動的環境における実時間視覚ベースSLAM)

July, 2021

Doctor of Philosophy (Engineering)

Yubao Liu
劉 玉宝

Toyohashi University of Technology

別紙４－１（課程博士（英文））

Date of Submission（month day, year）: July 7th, 2021

Department of Computer Science and Information Engineering	Student ID Number	D189302	Supervisors	Jun Miura Shigeru Kuriyama
Applicant's name	Yubao Liu			

Abstract（Doctor）

Title of Thesis	Real-time Visual Simultaneous Localization and Mapping under Dynamic Environments

Approx. 800 words

Visual simultaneous localization and mapping (SLAM) is a fundamental technology in robotics, augmented reality, computer vision, and self-driving. Visual SLAM mainly uses an onboard camera to perform camera ego-motion estimation and reconstruct the unstructured environment. There are two basic requirements for visual SLAM: robust tracking and real-time performance. Many visual SLAM algorithms use the rigid/static world assumption, limiting a wide deployment in the real world, such as populated scenes, as this strong assumption often results in poor tracking in dynamic environments.

Many studies have tried using semantic information to help visual SLAM reduce the influence of dynamic objects. The challenge is to trade off the tracking with semantic information and real-time performance. For example, Mask R-CNN can achieve good segmentation results and help improve tracking accuracy. However, it usually consumes much time, around 200ms. The efficiency of tracking is significantly limited by waiting for the segmentation results. We call such a model a *blocked model*. To achieve robust tracking while keeping the real-time performance, we proposed a *non-blocked model*, in which the tracking process is no longer blocked by waiting for the semantic results.

This thesis proposes four visual SLAM methods, which we explored better solutions to the problems mentioned above. The first two methods are developed based on the blocked model, while the others on the non-blocked model.

First, we present RTS-vSLAM, which achieves robust tracking by detecting and removing outliers on the dynamic objects using PSPNet and SegNet, and builds a static semantic map by excluding dynamic objects. However, the efficiency of the performance is limited due to the blocked model.

Second, to deal with dynamic non-pre-defined objects, in KMOP-vSLAM, we try to use k-means to segment all clusters or objects. We use OpenPose to judge which clusters belong to persons. However, k-means tend to over-segment, and the segmentation accuracy is worse than semantic segmentation methods. Time consumption is large due to the blocked model, too.

Third, we present RDS-SLAM, a real-time visual SLAM for dynamic environments, developed based on semantic segmentation and the non-blocked model. RDS-SLAM runs in real-time by evaluating the tracking and the segmentation thread in parallel. According to the semantic segmentation results, the motion information of features, represented as moving probability, is updated in the global map. Then we classify the features in the map into dynamic, static, and unknown using the moving probability. We use as many static features as possible in the data association and the bundle adjustment process to obtain robust tracking. We tested two semantic segmentation methods, SegNet and Mask R-CNN, using the TUM dynamic sequences. RDS-SLAM can run at 30 Hz with SegNet, while only at 15 Hz with Mask R-CNN to trade off the tracking robustness and the real-time performance.

Fourth, we present RDMO-SLAM, an extension of RDS-SLAM, which can run the Mask R-CNN version at 30 HZ with the help of dense optical flow. Since optical flow estimation is faster than Mask R-CNN segmentation, to get more segmented frames, we predict the semantic labels using dense optical flow and the segmented frames of Mask R-CNN. To cope with unknown dynamic objects, we also estimate the velocity of landmarks and then use it as another constraint to lower the influence of dynamic objects. These improvements make RDMO-SLAM with Mask R-CNN run at 30 Hz while keeping the tracking performance.

Contents

Chapter 1

Introduction

1.1 SLAM

The simultaneous localization and mapping (SLAM) [1] problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. SLAM can be implemented using different sensors, e.g., Camera, IMU, Lidar, and GPS. Generally, Lidar is much expensive than camera sensors. Visual SLAM (vSLAM) mainly using onboard camera sensors, such as mono, RGB-D, and stereo cameras. vSLAM has been a hot research topic in computer vision, augmented reality (AR), unmanned autonomous vehicles, and robotics. vSLAM [2] is a fundamental technology for estimating the pose of sensors and reconstructing structures in an unknown environment using onboard sensors, such as mono, RGB-D, and stereo cameras. vSLAM can be classified into feature based approaches, such as ORB-SLAM [3] and RGB-D SLAM [4], and direct approaches, such as LSD-SLAM [5] and DSO [6]. As we know, there is usually a strong assumption in vSLAM, the rigid scene assumption. vSLAM assumes that the camera is the only moving object. However, the camera is not the only moving object in the real environment, and this assumption may result in unstable tracking or even tracking failure. For instance, humans are dynamic objects in indoor environments. The motion of features or points on the dynamic objects is unknown and

Figure 1-1: Example of dynamic environments (TUM).

cannot be accurately estimated. Dynamic objects may influence feature matching and BA (bundle adjustment) and eventually resulting in non-robust pose estimation and map building.

As shown in Fig. 1-2, vSLAM by default cannot sense the motion of dynamic objects or dynamic features, e.g., m_{t-1}^{j} moved to m_{t}^{j}. Unfortunately, it still use the old map point m_{t-1}^{j} and its observed features $x_{t-1,j}$ and $x_{t,j}$ to estimate the pose using BA by minimizing the reprojection error for every selected features. The problem is that the newly observed feature, $x_{t,j}$, no longer matches m_{t-1}^{j} but matches a new landmark m_{t}^{j} that is unknown for vSLAM. This phenomenon can be somehow detected and reduced using geometric algorithms such as RANSAC (random sample consensus) [7] and BA if dynamic features move very fast. However, outliers cannot be efficiently detected if they move slowly or occupy a major part of the scene. As shown in Fig. 1-1, one person is walking slowly at the center of the image, and many features are detected on his T-shirts. vSLAM may mistakenly trust these features and result in non-robust pose estimation because these features are unstable.

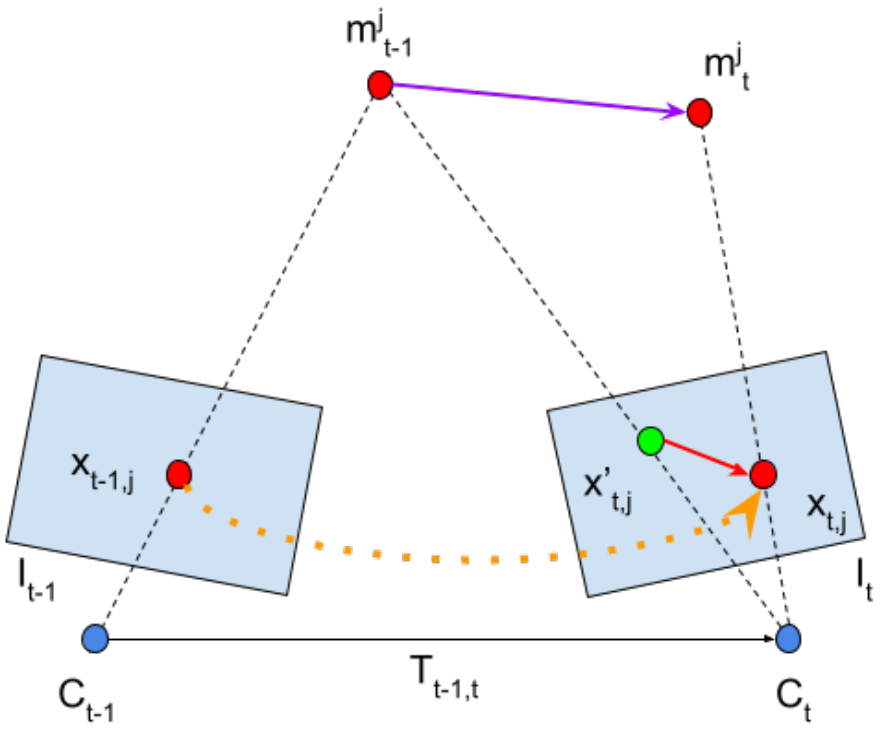


Figure 1-2: Rigid scene problem in vSLAM. The j-th map point m^j_{t-1} on a dynamic object matched with the feature $x_{t-1,j}$ in the previous image. Assume that this map point moved to a new position m^j_t and is observed as $x_{t,j}$ in the current image. $x'_{t,j}$ is the position if the feature is static. C_{t-1} and C_t are the camera centers of previous (I_{t-1}) and current (I_t) images.

1.2 Rigid Scene Assumption Problem

As shown in Fig. 1-2, given a 3D point in world coordinate $m^j_{t-1} = (x, y, z)^T \in \mathbb{R}^3$ at time $t-1$, the reprojection error of the predicted and the observed pixels at time t is defined as follows:

$$e_{t,j}(\xi) = x_{t,j} - \pi(T^w_t(\xi), m^j_{t-1}), \tag{1.1}$$

where, $x_{t,j} \in \mathbb{R}^2$ is the observed feature point; $T^w_t(\xi) = exp(\xi\hat{}) \in SE(3)$ is the pose of camera t under the world coordinate with $exp(.)$ as a mapping from $se(3)$ to $SE(3)$; $\xi \in \mathbb{R}^6$ is a 6D vector (3 for position and 3 for rotation), which is the target variable to be solved and optimized; π is a project function that projects a map point from the 3D space to the 2D image plane. In a static scene, $x_{t,j}$ should be in the position of $x'_{t,j}$ or very near position (influenced by the noise). vSLAM performs camera ego-

motion estimation by minimizing the reprojection error using the matched feature and landmark pairs. In practice, usually, BA is used to find an optimal solution using the error term Eq. (1.1) and the following cost function:

$$C = \sum_{t,j} \rho_h(e_{t,j}(\xi)^T \Omega_{t,j}^{-1} e_{t,j}(\xi)), \tag{1.2}$$

where a robust Huber kernel ρ_h is used to reduce the influence of spurious matching. For example, in ORB-SLAM3, g2o [8] is used to solve this BA problem. However, in dynamic environments, the observed and predicted positions may be different due to the movement of dynamic objects. For example, the old map point m_{t-1}^j moves to a new position/point m_t^j. By default, the traditional vSLAM cannot detect the motion and still use the old map point m_{t-1}^j to estimate the camera motion. If the motion of objects is considered, the error term should be defined as follows:

$$e_{t,j}(\xi) = x_{t,j} - \pi(T_t^w(\xi), m_t^j) \tag{1.3}$$

$$= x_{t,j} - \pi(T_t^w(\xi), H_{t-1}^t m_{t-1}^j), \tag{1.4}$$

where, H_{t-1}^t is the motion of the landmark m_{t-1}^j from the previous time $t-1$ to the current time. BA cannot optimize the camera pose correctly in dynamic environments due to the unknown motion H_{t-1}^t of landmarks. Usually, this operation will cause a non-robust camera pose estimation or tracking failure due to the large reprojection error.

To the best of our knowledge, there are two kinds of solutions. One solution [9,10] jointly optimizes the motion H of the object and the camera pose $T(\xi)$ using multi-object tracking by assuming the object is rigid and the points on it have the same or consistent motion. It has been reported that this assumption works in outdoor where the distances of objects are relatively large. However, this assumption does not hold for non-rigid objects, e.g., people in indoor environments. Besides, such methods are offline or not real-time because they use the blocked model.

Another solution is to detect outliers and remove them from tracking, which is

widely employed in the geometric and semantic-based approaches. In this case, the cost function is defined as follows:

$$C = \sum_{t,j} W_j \rho_h(e_{t,j}(\xi)^T \Omega_{t,j}^{-1} e_{t,j}(\xi)). \tag{1.5}$$

In some studies, W_j is assigned to 0 and 1 for dynamic and static points, respectively.

1.3 Research Goal

Although tremendous progress has been made in vSLAM, the scene rigidity assumption limits wide usage of visual SLAMs in the real-world environment of computer vision, smart robotics and augmented reality. To make SLAM more robust in dynamic environments, outliers on the dynamic objects, including unknown objects, need to be removed from tracking process. Our goal is to develop real-time and robust vSLAM that can run in real environment, especially for the populated environment. In addition, apply the developed vSLAM engine to AR, semantic mapping, localization task and some other applications. There are several requirements or subtasks: 1) real-time performance, 2) robust tracking in real world, and 3) acquire semantic information.

1.4 Contributions

To deal with the problem caused by the rigid world assumption, four algorithms are proposed as shown in Tab. 1.1. We first present a general system architecture in RTS-vSLAM, which runs localization and landmark mapping in *client* side and run semantic segmentation algorithms and semantic mapping in *server* side. However, semantic segmentation models can only deal with predefined objects. We try to segment all the clusters or objects using k-means in KMOP-SLAM together with the geometric check and human detection. However, such methods can not run in real-time due to waiting for the semantic result, such as semantic labels of semantic segmentation, clusters from k-means, and the key points of a person. In addition,

Table 1.1: Comparison of proposed works. We show the semantic models used, the average time consuming for tracking each frame, and the absolute tracking error (ATE) of the w/xyz dynamic scene of the TUM dataset.

Contributions	Pattern Recognition	Real-time Performance (ms)	Roboust Tracking (ATE of TUM)	Model
RTS-vSLAM	SegNet	147.47	0.014	Blocked Model
	PSPNet	214.95	0.012	
KMOP-vSLAM	OpenPose k-means	257.819	0.019	
RDS-SLAM	SegNet	22 - 30 (30HZ)	0.0571	Non-blocked Model
	Mask R-CNN	50 - 65 (15HZ)	0.0213	
RDMO-SLAM	Optical flow Mask R-CNN	22-35 (30HZ)	0.0226	

the tracking accuracy of KMOP-SLAM is not better than semantic-based solutions. To run vSLAM robustly while keeping the real-time nature, RDS-SLAM is proposed, which runs in 30 HZ using SegNet and 15 HZ using Mask R-CNN. To run in 30 HZ using Mask R-CNN and deal with the dynamic features in any object, RDMO-SLAM is proposed that uses semantic label prediction and feature velocity estimation with the aid of dense optical flow. We summarized the contributions as:

- RTS-vSLAM [11]: 1) improve tracking accuracy using PSPNet and SegNet using blocked model 2) build point cloud map and semantic map;
- KMOP-SLAM [12]: improve tracking accuracy using the geometric check, k-means, and OpenPose using blocked model;
- RDS-SLAM [13]: improve tracking accuracy using SegNet and Mask R-CNN using the non-blocked model;
- RDMO-SLAM [14]: improve tracking accuracy using Mask R-CNN and optical flow using the non-blocked model and semantic label prediction.

1.5 Thesis Organization

This thesis is organized as follows: We first investigated the related works in Chapter 2 and then describe the proposed methods in the following chapters, RTS-vSLAM

in Chapter 3, KMOP-vSLAM in Chapter 4, RDS-SLAM in Chapter 5, and RDMO-SLAM in Chapter 6. Finally, we conclude the thesis and discuss the future research works in Chapter 7.

Chapter 2

Related Work

In this chapter, we explore related works on vSLAM and state-of-the-art solutions to the rigid scene assumption in vSLAM under dynamic environments. We classify the methods into purely geometric, reconstruction, and semantic-based approaches. The semantic-based approaches leverage semantic information to detect and segment objects and remove outliers from tracking. Notably, these approaches may share some common ideas, such as the use of geometric checking.

2.1 Visual SLAM

vSLAM can be classified into feature based approaches, such as ORB-SLAM [3] and RGB-D SLAM [4], and direct approaches, such as LSD-SLAM [5] and DSO [6]. Feature-based methods rely on salient point matching and can only perform a sparse reconstruction. Parallel tracking and mapping (PTAM) [15] that implements a keyframe-based monocular SLAM system on a cell phone is a promising platform for hand-held AR. ORB-SLAM [3], a monocular vSLAM that estimates camera ego-motion by matching ORB [16] features extends the versatility of PTAM to environments that are intractable for PTAM. Based on ORB-SLAM, ORB-SLAM2 [17], a complete SLAM system for monocular, stereo, and RGB-D camera was presented, which can work in real-time in various environments. Carlos et al. proposed the latest version of ORB-SLAM, ORB-SLAM3 [18], which tightly integrates visual and

inertial information and adds a multiple map system (ATLAS [19]).

Apart from feature-based methods, many direct vSLAM approaches [5, 6, 20, 21], which can estimate, in principle, a completely dense reconstruction by the direct minimizing of the photometric error, have also been proposed. For example, Kerl et al. proposed a dense visual SLAM method, DVO (Dense Visual SLAM) [21], using an RGB-D camera, which minimizes both photometric and depth error over all pixels.

However, rigid scene assumption is a common problem for both the feature-based and the direct methods. Detecting and handling outliers in real-time is challenging in vSLAM. Although there are some strategies, such as selecting relative good features and RANSAC-based checking, in some vSLAMs, e.g., ORB-SLAM3. However, they are still not well suitable for dynamic environments.

2.2 Pattern Recognition

Many pattern recognition algorithms are proposed with the development of machine learning and deep learning, e.g., semantic segmentation, object recognization, and unsupervised segmentation. These methods can be used to help SLAM, e.g., to support semantic information or help to improve tracking accuracy. Semantic segmentation algorithms, e.g., SegNet [22], PSPNet [23], and Mask R-CNN [24] are widely used in computer vision and robotics, which can segment each image at the pixel level. We can judge which pixel belonging to which object in SLAM. The evaluating time of these models varies and some models are very heavy. To evaluate semantic segmentation algorithms in real-time is a big challenge when integrating them into SLAM as the real-time nature of SLAM. As we know, semantic segmentation algorithms only can identify pre-defined objects. Unsupervised segmentation algorithms, e.g., k-means [25] can segment any clusters or objects. However, k-means cannot segment objects as accurately as semantic segmentation, and no semantic label is afforded by k-means. Some object recognization algorithms can run in real-time or very fast, e.g., YOLO [26] and Yolact [27]. However, the output of them is only bonding boxes. The label of each pixel cannot be judged only using the bounding box. Some algorithms

can detect the key points of the people, e.g., OpenPose [28]. It is difficult to know the semantic label and region of objects only using them because they can only output few key points.

2.3 Visual SLAM under Dynamic Environments

2.3.1 Geometric-based Approaches

Li et al. [29] proposed a depth edge-based RGB-D SLAM system for dynamic environments based on the frame-to-keyframe registration, which only uses weighted depth edge points. Sun et al. [30] proposed a novel online RGB-D data-based motion removal approach that uses optical flow. It is integrated with the front end of an RGB-D SLAM system, acting as a preprocessing stage to filter data associated with dynamic objects. They also proposed a monocular vSLAM algorithm [31] that uses optical flow to improve tracking performance with a monocular camera in dynamic environments. Also, they integrated their method into ORB-SLAM. However, their methods have some limitations. For example, the threshold used to distinguish dynamic points is set to a fixed value, which may not be an optimal value for some sequences. Kim et al. [32] proposed an IMU-based solution. They classified the features into dynamic and static using the IMU rotation component between two consecutive images. However, for many use cases, it is desirable to improve the accuracy of pose tracking and map building using only a single camera. Besides, IMU has drift and accumulated errors over time. Sun et al. [33] classified pixels using the segmentation of quantized depth images and calculated the difference in intensity between consecutive RGB images. Tan et al. [34] proposed a novel online keyframe representation and updating method to adaptively model the dynamic environments, where an appearance or a structure change could be effectively detected and handled. Although geometric-based methods can eliminate outliers to some extent, there is room for further optimization of tracking performance using semantic information.

2.3.2 Reconstruction-based Approaches

Visual odometry and scene flow (VO-SF) [35], an odometry-based method designed for dynamic scenes proposed by Jaimez et al., combines visual odometry, k-means, and scene flow and reconstructs a 3D model of the rigid scene. Co-Fusion [36] proposed an approach to track and reconstruct multiple moving objects using SharpMask [37]. BaMVO [38] proposed a background model-based visual odometry. StaticFusion [39], a method for dense RGB-D SLAM proposed by Raluca et al., tried to address the rigid scene assumption by jointly estimating the motion of an RGB-D camera and segmenting the scene into static and dynamic parts. StaticFusion is conceptually related to BaMVO but uses a frame-model alignment instead of a multi-frame strategy. Similar to ElasticFusion [40], camera tracking is performed by aligning incoming frames with a dense surfel-based model of the environment. A background model that fuses only the static elements by decoupling the static and dynamic parts is built. K-means is used to segment geometric clusters in StaticFusion, and it is difficult to find the optimal K value for a specific scene. This problem also exists in other k-means-based algorithms, such as KMOP-vSLAM [12]. Also, StaticFusion assumes each cluster is a rigid body to reduce the overall computational complexity, and then solves the static/dynamic segmentation problem cluster-wise as opposed to pixel-wise. Moving people are not rigid bodies in many cases. We do not use such an assumption because we focus on improving the tracking accuracy by eliminating outliers both on rigid objects and dynamic objects in real-time rather than focusing on building a conservative reconstruction of the static structures of the scene.

In this study, the camera pose is estimated using sparse ORB features because it is usually more lightweight than the dense RGB-D SLAM, and we do not build the dense surfel-based model.

2.3.3 Semantic-based Approaches

DynaSLAM [41], based on ORB-SLAM2 and Mask R-CNN has capabilities of dynamic object detection and background inpainting, which can detect dynamic objects

either by multiview geometry, deep learning, or both and then reconstructs frame backgrounds occluded by dynamic objects using a rigid scene map. DP-SLAM [42] combines the results of geometry constraints and Mask R-CNN to track the dynamic key points in a Bayesian probability estimation framework. DP-SLAM was integrated into the front-end of the ORB-SLAM2 to inpaint frame background occluded by the detected dynamic objects. KMOP-vSLAM [12], also implemented on ORB-SLAM2, has capabilities of unsupervised learning segmentation (k-means [25]) and human detection (OpenPose [28]) for robust tracking in dynamic environments. Outliers belonging to dynamic objects are detected and eliminated from tracking. One problem is that the number of clusters of k-means is given manually and it may not be optimal for the current environment. DS-SLAM [43], based on ORB-SLAM2 and SegNet [22], uses a moving consistency check to reduce the impact of dynamic objects by assuming that feature points on people are most likely to be outliers. Detect-SLAM [44], based on ORB-SLAM2 and SSD [45], classifies keypoints into four states: low-confidence static, high-confidence static, low-confidence dynamic, and high-confidence dynamic. It only detects keyframes to save time and then insert the keyframes into the local map and update the moving probability into the local map. DM-SLAM [46], also based on ORB-SLAM2, employs Mask R-CNN, optical flow, and epipolar constraints to judge outliers. It uses features in dynamic objects if they are not moving very fast to reduce the feature-scarce cases that may happen by eliminating the features on dynamic objects. Fan et al. [47] proposed a novel semantic SLAM system with a more accurate point cloud map in dynamic environments by exploiting ORB-SLAM2 and BlizNet [48].

Most existing algorithms operating in complex dynamic environments simplify problems by eliminating dynamic objects from tracking or tracking them separately. However, VDO-SLAM [9] presented a novel formulation to model dynamic scenes in a unified estimation framework over robot poses, static and dynamic 3D points, and object motions. DynaSLAM II [10] is a similar work with VOD-SLAM that tracks multiple rigid objects such as cars and bicycles. According to the data provided in their papers, DynaSLAM II is a little faster and more robust than DVO-SLAM

if not considering the time complexity of semantic segmentation. However, these methods only work for rigid objects and neither of them is suitable for indoor dynamic environments where people are the major dynamic objects. For example, In the dynamic scene of the TUM [49] dataset, people change their shape sometimes by standing or sitting. Besides, these methods are not real-time because the semantic segmentation and optical flow information need to be prepared beforehand.

All the methods that use the blocked model wait for the semantic results of each frame or keyframe before estimating the camera pose, thereby resulting in their processing speed being limited by the segmentation method used. To further clarify this, we compared the tracking performance and time complexity with state-of-the-art works.

Chapter 3

RTS-vSLAM

Real-time Visual Semantic Tracking and Mapping (RTS-vSLAM) is implemented based on RGB-D-based ORB-SLAM2, which uses both semantic information and geometric methods to detect moving objects and outlier features. RTS-vSLAM adopts a novel generic system architecture that enables the use of many kinds of semantic segmentation methods on the remote server and can be employed on embedded system devices, head-mounted devices, and low-performance robots, where GPUs are not mounted on, via requesting service from the server. It also generates many kinds of maps to support navigation and more complex tasks in real-time. The proposed system is evaluated using the TUM RGB-D dataset and compared tracking errors with the state-of-the-art vSLAM methods under dynamic environments. RTS-vSLAM outperforms the others in tracking accuracy in a highly dynamic scenario and reduces the time delays greatly.

Many solutions have been proposed for the above problems. Geometry-based methods, e.g. IMU and RANSAC [7], are the main ones in the past. In recent years, semantic segmentation based methods have also been tried. In 2018, DS-SLAM [43] and DynaSLAM [41] were proposed. However, both of them use only one kind of specific semantic segmentation method. Furthermore, Semantic segmentation is time-consuming and needs a lot of computing resources, which limits the wide employment on many embedded devices and mobile robots in real-world application.

RTS-vSLAM proposes a generic server-client architecture. The server has remote

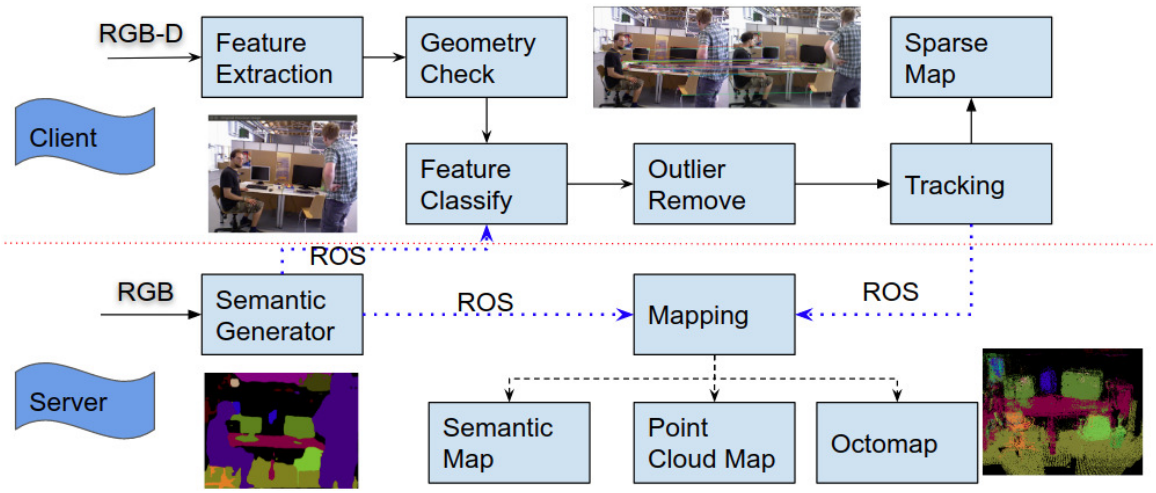


Figure 3-1: System overview of RTS-vSLAM

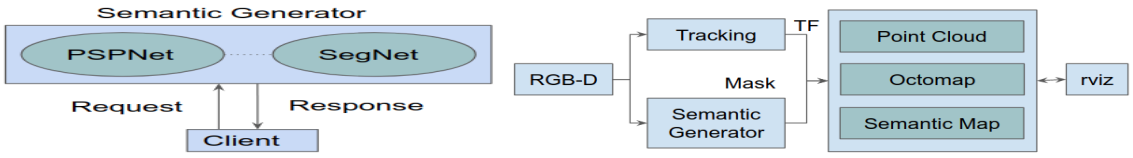
computing and monitoring ability where semantic segmentation and mapping modules are employed. Users can switch semantic segmentation methods according to a specific environments and monitor the map remotely. The client-side devices only need to do the tracking, where the state-of-the-art vSLAM systems can be employed.

The main contributions of RTS-vSLAM are: 1) A generic semantic SLAM system, RTS-vSLAM, is proposed, which is implemented based on ORB-SLAM2 and robust to the dynamic environment. It includes novel feature clustering and tracking algorithms to reduce the influence of dynamic objects using object information.

2) Server and Client architecture. Mapping and semantic modules are deployed on the server, which makes it possible to deploy the system on devices which lack computing resources. In principle, many kinds of semantic segmentation methods can be supported on the server and different kinds of maps built, where dynamic objects are mostly removed.

3.1 System Overview

Fig. 3-1 shows an overview of the proposed system. There are two parts in the architecture: **server** and **client** communicated with ROS(labeled with blue dot lines). **Server** is designed to deal with computing complex tasks: semantic segmentation and mapping. **Client** is only for tracking. Server-end can be employed in remote PCs



(a) Semantic architecture (b) Mapping architecture

Figure 3-2: Architecture overview of server

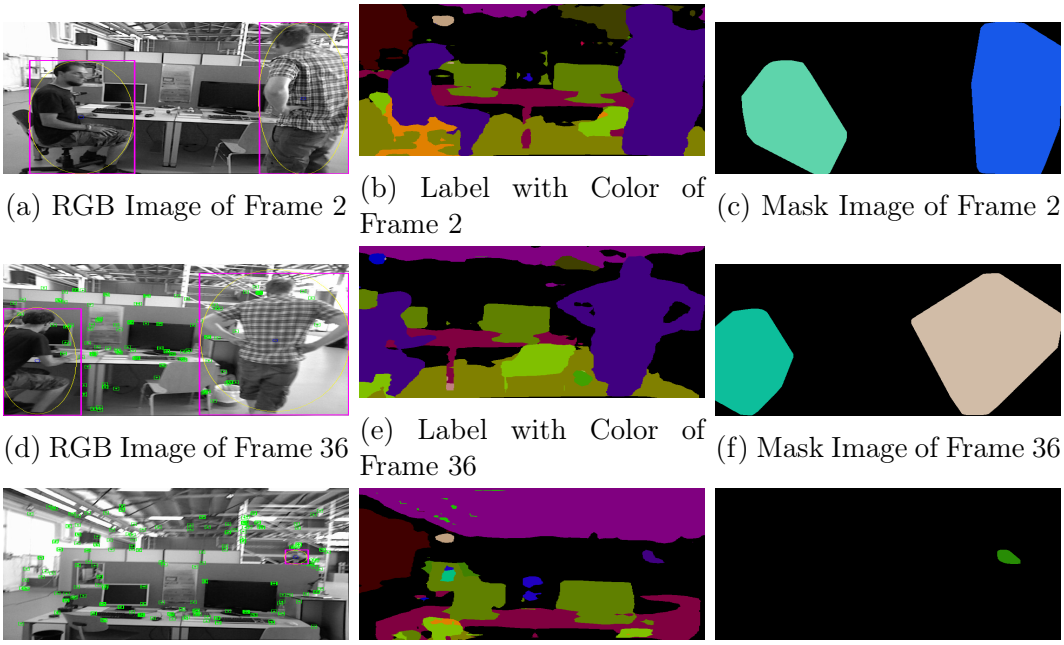
that have higher computing capability. We built three kinds of maps on the server, which are labeled by dark dot lines.

First, the RGB channels pass through the Semantic Generator, where object labels are generated (server) and the Feature Extraction module(client) where ORB features are extracted. Secondly, geometric moving consistent check is used to select good features by removing the feature points that are not consistent with the main moving trend. After that, the feature distribution is calculated by injecting semantic labels into feature points, giving each feature an object name, which is useful to remove dynamic features. Thirdly, these feature points are classified using prior knowledge. Next, the outliers are removed in the outlier removal module. Finally, the camera pose is estimated in the tracking module by matching feature points where dynamic features are not included.

3.2 Server

3.2.1 Semantic Generator

Different semantic segmentation networks can be trained using different datasets. e.g. PASCAL VOC [50], Cityscapes [51], SUN RGB-D [52], and ADE20K [53] datasets. PSPNet [23] (trained with ADE20K) and SegNet (trained with PASCAL VOC) were used, as shown in Fig. 3-2(a). The semantic module accepts the segmentation request and responds to the client, the semantic mask. Segment results are shown in Fig. 3-3(b), (e) and (h). Each person was segmented and a bounding box add to each person using contour detection and labeling, as shown in (a), (d) and (g). A mask was generated to remove dynamic features as shown in (c),(f) and (i).



(a) RGB Image of Frame 2 (b) Label with Color of Frame 2 (c) Mask Image of Frame 2

(d) RGB Image of Frame 36 (e) Label with Color of Frame 36 (f) Mask Image of Frame 36

(g) RGB Image of Frame 342 (h) Label with Color of Frame 342 (i) Mask Image of Frame 342

Figure 3-3: Segmentation result and mask of dynamic objects using PSPNet

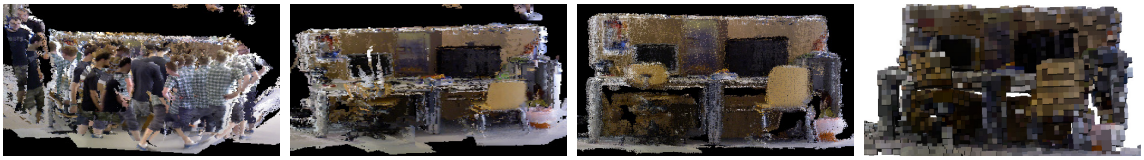
3.2.2 Mapping

An attempt was made to establish a semantic map, point cloud map, and octomap [54] in real-time on the server-end. The mapping architecture is shown in Fig. 3-2(b), where maps were generated using Odom (TF) and the semantic label obtained from the semantic generator. Finally, the ROS rviz[1] tool was used to visualize the map. Fig. 3-4 shows the semantic map result using PSPNet. The color is the same as the color in Fig. 3-3 (b),(e), and (h).

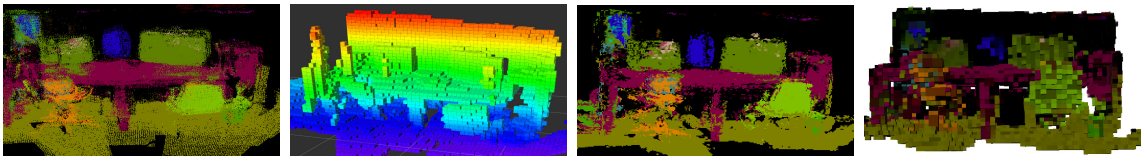
3.3 Client

The main idea of the RTS-vSLAM client-end algorithm is shown in Alg.1. Firstly, ORB features S_t were extracted in the latest frame F_t. Then, relatively good features were selected using the geometry moving consistently check algorithm. However, the distribution of these features was not known without the semantic label. **Objects**

[1]http://wiki.ros.org/rviz



(a) Point Cloud Map using Original ORB SLAM2 (b) Point Cloud Map using RTS-vSLAM online (c) Filtered Result of Point Cloud Map using RTS-vSLAM (d) Filtered Result of Octomap using RTS-vSLAM



(e) Semantic map online result (f) Octomap online result (g) Filtered semantic map (h) Filtered semantic Octomap

Figure 3-4: Result of mapping using TUM f3_walk_xyz dataset and PSPNet

Algorithm 1 RTS-vSLAM Algorithm Overview

Require: Previous and Current Image: F_{t-1}, F_t
Previous Feature Set: S_{t-1}
Pixel-wise segmentation mask for F_t: L_t
Ensure: Inlier Feature Set of F_t: S_t
Camera pose (TF): (R, T)
1: $S_t = FeatureExtract(F_t)$
2: $GeometryMovingCheck(F_{t-1}, F_t, S_t)$
3: $C_t^s, C_t^d, C_t^u = objectCluster(Dataset)$
4: $S_t^d, S_t^s, S_t^u = featureCluster(F_t, C_t^s, C_t^d, S_t)$
5: $RemovePrioriDynamicFeature(S_t^d, S_t)$
6: $(R, T, S_t) = Tracking(S_{t-1}, S_t^s, S_t^u)$
7: **return** $TF(R, T), S_t$

were placed into three classes: 1) static object set C_t^s (objects with high probability to keep static), 2) dynamic object set C_t^d and 3) unknown object set C_t^u (e.g. objects not trained and objects which cannot be clearly judged as static or dynamic), which is done only in the initialize stage, not in the main loop. Then **features** were classified into three sets according to their semantic label: 1) static feature set S_t^s, i.e. features belong to the a static object, 2) dynamic feature set S_t^d and 3) unknown feature set S^u. Features in dynamic feature set S_t^d were removed because the features belonging to highly dynamic objects (e.g. person) are outliers in high probability. Finally, these features were matched to calculate the camera ego-motion using the object-based

Algorithm 2 Feature Cluster Algorithm

Require: Current Frame: F_t;
Object Cluster: C_t^s, C_t^d;
Feature points in F_t: S_t;
Ensure: (Dynamic, Static, Unknown) feature set in F_t:
$(S_t^d,\ S_t^s,\ S_t^u)$

for $p_t \in P_t$ **do**
  if semantic label $l_t^{p_t} \in C_t^d$ **then**
    $S_t^d = S_t^d + p_t$
  else if $l_t^{p_t} \in C_t^s$ **then**
    $S_t^s = S_t^s + p_t$
  else
    $S_t^u = S_t^u + p_t$
  end if
end for
return $(S_t^d,\ S_t^s,\ S_t^u)$

tracking algorithm we proposed.

3.3.1 Feature Cluster Algorithm

Feature cluster algorithm is shown in Alg.2, For each feature point p_t in features S_t of latest image F_t, Object names were checked as to where they belonged to. Features were classified into a dynamic feature set S_t^d if their semantic label was a dynamic object defined in advance. Similarly, static feature set S_t^s and unknown feature set S_t^u were classified.

3.3.2 Tracking Algorithm

The tracking algorithm is shown in Alg.3. The matching algorithm of original ORB SLAM2 was customized so as to use rigid features as far as possible. Firstly, static features set S_t^s in current frame F_t were matched with features S_{t-1} in previous frame. $S_t^s + S_t^u$ was matched with S_{t-1} if the matched number in the previous step was less than a given threshold τ (set to 20 in the experiment, the same as the original ORB SLAM2). Finally, camera ego-motion is calculated using matched features M with the help of pose graph optimization.

Algorithm 3 Tracking Algorithm

Require: Feature set of previous image: S_{t-1}
Feature set of current image: S_t^s, S_t^u
Ensure: Inlier feature points: S_t
Camera pose (TF): (R, T)

if $FeatureMatch(S_{t-1}, S_t^s, M) < \tau$ **then**
  if $FeatureMatch(S_{t-1}, S_t^s + S_t^u, M) < \tau$ **then**
    return *continue*
  end if
end if
$(R, T, S_t) = CalculateCameraEgoMotion(M)$
return $TF(R, T), S_t$

Algorithm 4 Geometry Consistency Check Algorithm

Require: Feature Points of Current Frame: S_t
Previous and Current Image: F_{t-1}, F_t
Ensure: Feature Points: S_t

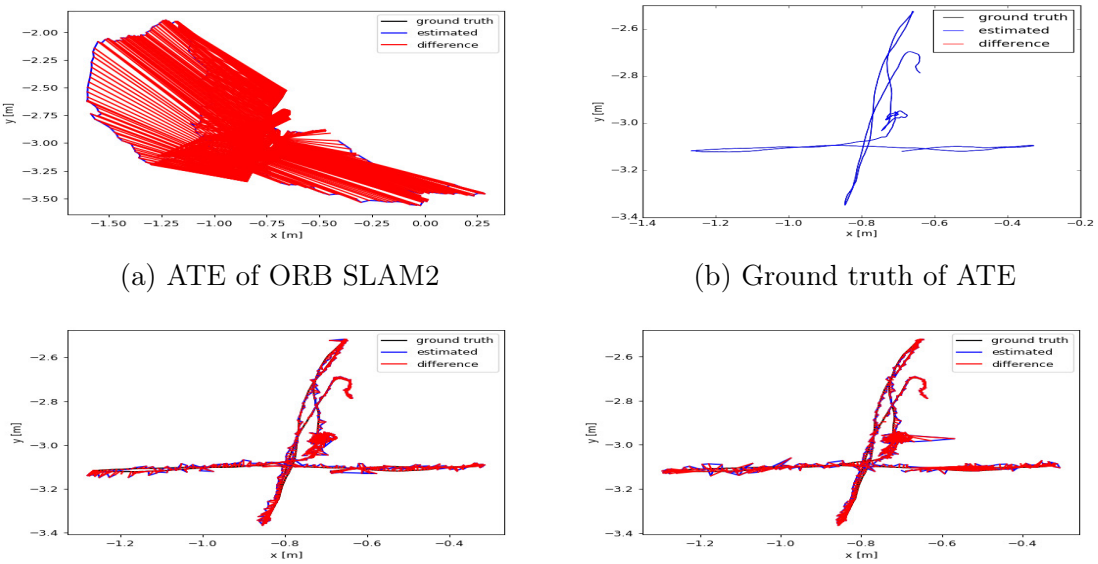
$calcOpticalFlowPyrLK(F_t, F_{t-1}, S_t, S'_{t-1})$
for $p_t \in S_t; p'_t \in S'_{t-1}$ **do**
  Remove the feature p_t if near to the image boarder
  for $\forall dx, \forall dy \in \{-1, 0, 1, -1, 0, 1, -1, 0, 1\}$ **do**
    $sum+ = F_t[p_t + [dx, dy]^T] - F_{t-1}[p_t^{'} + [dx, dy]^T]$
  end for
  if $sum > \tau$ **then**
    $S_t = S_t - p_t$
  end if
end for
return S_t

3.3.3 Geometry Consistency Check Algorithm

The geometry moving consistent moving check algorithm used is shown in Alg.4. Sparse optical flow and cross-correlation were used to check moving consistency. The window size of neighbor was set to $3 * 3$. τ is a hyper-parameter that was given statistically.

3.4 Experimental Results

The proposed system was evaluated using a public dataset, TUM RGB-D [49], and compared to the state-of-the-art vSLAM systems under dynamic environments, using,



(a) ATE of ORB SLAM2

(b) Ground truth of ATE

(c) ATE of RTS-vSLAM (PSPNet)

(d) ATE of RTS-vSLAM (SegNet)

Figure 3-5: Tracking error against to ORB SLAM2 using TUM f3/walk_xyz

when possible results, published in the original papers. To evaluate the semantic generator module, the tracking performance was evaluated using both PSPNet and SegNet. Furthermore, the system was compared against to the ORB-SLAM2 to quantify the improvement offered by the proposal in highly dynamic scenarios.

The TUM RGB-D dataset contains color and depth images along the ground-truth trajectory of the sensor. In the sequence named "*fr3/walking_ **" (labeled as f3/w_*), two people walk through an office. This is intended to evaluate the robustness of vSLAM using quickly moving dynamic objects in large parts of a visible scene. Four types of camera motion are included in *walking* data sequences 1) "*xyz*", the Asus Xtion sensor is manually moved along three directions (xyz); 2) "*static*", where the camera is kept in place manually; 3) "*halfsphere*", where camera is moved on a small half sphere of approximately one meter diameter; 4) "*rpy*", where camera is rotated along the principal axes *(roll-pitch-yaw)*.

The error in the estimated trajectory was calculated by comparing it with the ground truth, using two prominent methods: Absolute Trajectory Error (**ATE**) and Relative Pose Error (**RPE**) [49], which are well-suited for measuring the performance of the vSLAM. The proposal was also evaluated against other vSLAMs under dynamic

Table 3.1: Comparison of ATE [m] against the state-of-the-art vSLAMs

Sequences	ORB SLAM2		DS-SLAM (SegNet)			RTS-vSLAM (SegNet)			RTS-vSLAM (PSPNet)		
	RMSE	Median	RMSE	Median	Improve	RMSE	Median	Improve	RMSE	Median	Improve
f3/w_xyz	0.7521	0.5857	0.0247	0.0151	96.71%	0.019	0.013	97.47%	**0.016**	**0.012**	**97.87%**
f3/w_static	0.3900	0.3087	0.0081	0.0067	97.91%	0.01	0.007	97.44%	**0.007**	**0.006**	**98.21%**
f3/w_rpy	0.8705	0.7059	**0.4442**	**0.2835**	**48.97%**	0.198	0.036	77.25%	**0.043**	**0.024**	**95.06%**
f3/w_half	0.4863	0.3964	0.0303	0.0222	93.76%	0.039	0.032	91.98%	**0.03**	**0.022**	**93.83%**

Table 3.2: Comparison of the median value [m] of ATE(RMSE) against the state-of-the-art vSLAMs in dynamic scenes

Sequences	ORB SLAM2	Depth Edge SLAM	Motion Removal DVO-SLAM	Motion Segmentation DSLAM	DS-SLAM (SegNet)	DynaSLAM (N+G+BI)	RTS-vSLAM (SegNet)	RTS-vSLAM (PSPNet)
f3/w_xyz	0.5857	0.060	0.093	0.040	0.0151	0.015	0.014	**0.012**
f3/w_static	0.3087	0.026	0.066	0.024	0.0067	0.007	0.007	**0.006**
f3/w_rpy	0.7059	0.179	0.133	0.076	0.2835	0.136	0.123	**0.024**
f3/w_half	0.3964	0.043	0.125	0.055	0.0222	0.029	0.027	**0.022**

environment by comparing the Root Mean Squared Error (**RMSE**), and median values, defined in [49], and the **improvement** (defined as Eq.3.1) evaluated against ORB SLAM2 using RMSE of ATE and RPE. Each sequence was ran at least five times as dynamic objects are prone to increase the non-deterministic effect.

$$Improve = \frac{RMSE(ORBSLAM2) - RMSE(Proposed)}{RMSE(ORBSLAM2)} \tag{3.1}$$

3.4.1 Evaluation on TMU Dataset

The proposed system was compared to ORB SLAM2 and DS-SLAM in detail as shown in Tab. 3.1 and the tracking error visualized using f3/walk_xyz dataset as an example, shown in Fig. 3-5. Fig. 3-5(a) shows the result of the original ORB SLAM2. The tracking error (red line) is very large; (c) and (d) show the result of RTS-vSLAM where error is largely reduced. RMSE of ATE (median error) was also compared with other dynamic vSLAM algorithms, as shown in Tab. 3.2. RPE error was also compared with other methods, as shown in Tab. 3.3 and Tab. 3.4. RTS-vSLAM, achieved good performance in tracking against other vSLAMs in dynamic environments. PSPNet version is a little better than SegNet in RTS-vSLAM because its segmentation accuracy is a little better than SegNet.

Table 3.3: Comparison of the result of metric translation (RPE) [m] against the state-of-the-art vSLAMS in dynamic scenes

Sequences	ORB SLAM2		DS-SLAM (SegNet)			RTS-vSLAM (SegNet)			RTS-vSLAM (PSPNet)		
	RMSE	Median	RMSE	Median	Improve	RMSE	Median	Improve	RMSE	Median	Improve
f3/w_xyz	0.412	0.246	0.033	0.018	91.93%	0.025	0.017	93.94%	**0.02**	**0.016**	**95.15%**
f3/w_static	0.2162	0.0155	0.0102	0.0082	95.27%	0.012	0.009	94.45%	**0.009**	**0.007**	**95.84%**
f3/w_rpy	0.4249	0.1487	**0.1503**	**0.0457**	**64.64%**	0.141	0.053	65.88%	**0.06**	**0.031**	**85.88%**
f3/w_half	0.355	0.0774	0.0297	0.0226	91.62%	0.037	0.025	89.58%	**0.028**	**0.022**	**92.11%**

Table 3.4: Comparison of the result of metric rotation (RPE) [deg] against the state-of-the-art SLAMS in dynamic scenes

Sequences	ORB SLAM2		DS-SLAM (SegNet)			RTS-vSLAM (SegNet)			RTS-vSLAM (PSPNet)		
	RMSE	Median	RMSE	Median	Improve	RMSE	Median	Improve	RMSE	Median	Improve
f3/w_xyz	7.7432	4.534	0.8266	0.4192	89.33%	0.691	0.425	91.08%	**0.607**	**0.394**	**92.16%**
f3/w_static	3.8958	0.3571	0.269	0.2259	93.09%	0.295	0.242	92.43%	**0.25**	**0.208**	**93.58%**
f3/w_rpy	8.0802	2.7828	**3.0042**	**0.9902**	**62.82%**	2.757	1.105	65.88%	**1.316**	**0.704**	**83.71%**
f3/w_half	7.3744	1.8143	0.8142	0.6217	88.96%	0.895	0.636	87.86%	**0.775**	**0.61**	**89.49%**

Table 3.5: Evaluation of time taken (ms)

Time f3/w_rpy	Server Semantic Generator	Server Mapping	Client Low Cost Tracking	Client Geometry Check	Client ORB Feature	Client Feature Classify	Total Time Each Frame
PSPNet	178.08	46.19	2.65	52.43	89.85	0.16	214.95
SegNet	27.77	18.71	2.37	35.34	73.76	0.12	147.47

3.4.2 Result of Mapping

Fig. 3-4(a) is the original point cloud map created by ORB SLAM2 using TUM f3_walk_xyz dataset. The map is very dirty due to the influence of dynamic objects. (b) is the point cloud map created in real-time use RTS-vSLAM where the walking person is almost removed from the scene. However, we cannot perfectly remove all of them due to the occasional wrongly semantic segmentation or inaccurately estimated camera ego-motion. (c) is a filtered point cloud map where noise (e.g. the remaining parts of person) is mostly removed. (d) is the octomap corresponding to (c). We also created semantic map (e) and octomap (f) in Real-time. (g) is filtered results of (e) after removing noise and (h) is the filtered octomap where a semantic label is injected.

3.4.3 Computation Time Analysis

To complete the evaluation of the system, Intel i7 CPU and GeForce GTX 1080 Ti GPU (server-end only) were used to evaluate the average computational time for each module as shown in Tab. 3.5. DynaSLAM reported their timing performance: Mask-RCNN (195ms) on an Nvidia Tesla M40GPU, Low-Cost tracking (1.59ms), Multi-view Geometry (235.98ms) and Background Inpainting (183.56 ms) in TUM f3/walk_rpy dataset. The delay was caused by region growth algorithm in multi-view geometry stage and background inpainting. In RTS-vSLAM, Semantic segmentation, mapping, and tracking (ORB Feature extraction, descriptor computing and geometric check) running in parallel in different thread or different machines reduced the delay.

3.5 Conclusions

A semantic SLAM system, RTS-vSLAM, is presented, which is implemented based on ORB SLAM2 and robust to dynamic environments. Efficient tracking algorithms have been developed using object information and geometric method to reduce the tracking error caused by dynamic objects, and better results are achieved compared to the state-of-the-art vSLAM methods using a dynamic environment dataset (TUM dataset). In addition, the Server and Client architecture makes it possible to employ this system on client-end devices by performing computation-heavy parts, semantic segmentation, and mapping, at the server-end rather than mobile robots. This also makes it possible for the ORB feature extraction and descriptor computation to run in parallel with the semantic segmentation, thereby greatly reducing the delay caused by semantic label prediction. At the same time, static maps are generated and visualized on the server remotely in real-time.

Chapter 4

KMOP-vSLAM

RTS-vSLAM use semantic segmentation algorithms to deal with the rigid world assumption. However, it can only deal with the dynamic points on the predefined objects. To address this challenge, we present a novel real-time visual SLAM system, KMOP-vSLAM, which adds the capability of unsupervised learning segmentation and human detection to reduce the drift error of tracking in indoor dynamic environments. An efficient geometric outlier detection method is proposed, using dynamic information of the previous frames as well as a novel probability model to judge moving objects with the help of geometric constraints and human detection. Outlier features belonging to moving objects are largely detected and removed from tracking. The well-known dataset, TUM, is used to evaluate tracking errors in dynamic scenes where people are walking around. Our approach yields a significantly lower trajectory error compared to state-of-the-art visual SLAMs using an RGB-D camera.

4.1 Problem Description and Contributions

Geometric-based solutions [33,55,56] and semantic label-based methods [41,43,46,47] that remove outliers using semantic labeling of all pixels are proposed. Semantic segmentation-based methods achieved better tracking performance than purely using geometric methods. However, SLAM techniques localize build a map of an unknown environment and localize the sensor in the map with a strong focus on

(a) RGB (b) Segmentation (c) Feature

Figure 4-1: Dynamic scene. (b) is the segmentation result of K-means, (c) the feature distribution using geometric constraint where epipolar lines are shown. Red dots are outliers and blue dots are other features.

real-time operation [17]. Pixel-level semantic segmentation only can segment known objects that have been trained in advance, and it is time-consuming and computing resource-consuming task. Unsupervised learning segmentation methods and geometric constraints are used in [57], which can segment the objects or regions (collectively referred as objects in the following text) in unknown scene and detect moving objects based on geometric errors. This basic idea is followed in this paper to segment the objects that not trained in advance (e.g., Fig. 4-1(b)). This combination can make SLAM suitable to unknown environments and operate in real-time. However, the segmentation result is sometimes inaccurate (Fig. 4-1(b)) and it is hard to detect moving people if they wear uniformly colored clothes because only a few features can be detected on the edge of the body (Fig. 4-1 (c) (*left person*)).

We try to improve the accuracy of moving object detection using unsupervised learning segmentation (e.g., K-means), human detection, and geometric constraint. K-means can segment unknown environment into K objects, but it is difficult to know where the people are, e.g., person in the left in Fig. 4-1 using only geometric information. Human detection matched can locate people, but, it cannot detect other movable objects such as the chair someone is pulling or pushing. The dynamic features on these objects is detected and removed using K-means and geometric methods.

We proposed a novel dynamic visual SLAM using K-means and OpenPose (KMOP-vSLAM), built on RGB-D based ORB SLAM2 [17], for dynamic indoor environments. The main contributions are:

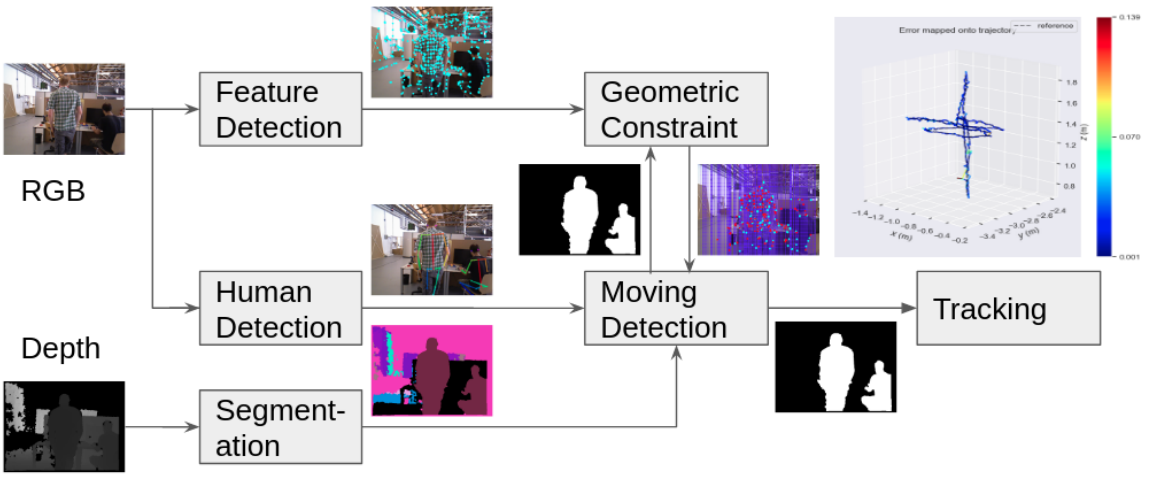


Figure 4-2: System overview of KMOP-vSLAM. RGB images pass through Feature Detection and Human Detection to extract ORB features and locate the person. Depth images are used to segment objects. Ouliers are removed using segmentation results, person position and geometric methods. The mask of dynamic objects is used for the next iteration and pose estimation.

1) a novel moving object detection method that is based on a probability model leveraging K-means, human detection and geometric methods;

2) an efficient geometric outliers detection method that uses moving object information of previous frame and a good matched features selection algorithm;

3) an unsupervised learning segmentation method is used to segment the objects including the unknown objects;

4) the integration of all of the above techniques to detect and remove dynamic objects, including people, more accurately in dynamic indoor environments in real-time.

4.2 System Implementation

Our system incorporates these parts: unsupervised learning segmentation, person detection, multi-view geometric check, and probability-based moving object detection algorithm.

4.2.1 System Overview

Fig. 4-2 shows the whole architecture of the proposed system. Three modules run in parallel: Feature Detection, Segmentation, and Human Detection. An RGB image passes through the Human Detection module to locate the person and passes through the Feature Detection module to extract ORB features which are used for geometrically-based moving object detection and camera ego-motion estimation. The depth image passes through segmentation and get the pixel label using K-means. Next, the moving object is detected using the result of the Geometric Constraint and Human Detection. Finally, camera ego-motion is estimated using rigid features after removing outliers on the moving objects.

4.2.2 Unsupervised Learning Segmentation

K-means is used to segment depth images (e.g., Figs. 4-3(c) and (d)) to generate objects. Morphology filter is used to fill invalid holes in the depth image. In our experiment, in order to compare our proposal with existing similar work [57], 10 clusters (same as [57]) are used and the results are shown in Figs. 4-3(e) and (f).

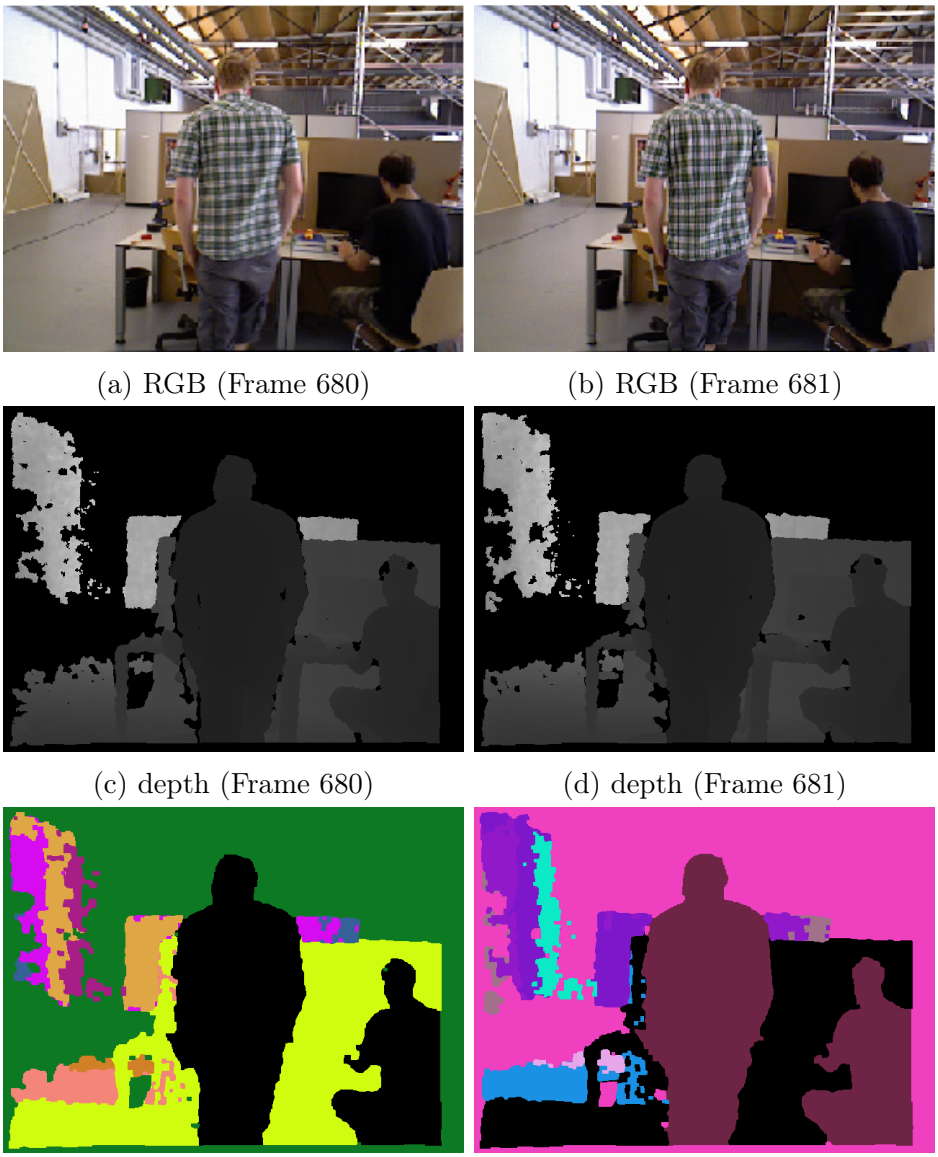
4.2.3 Multi-view Geometry Algorithm

Geometric Error Function

Assume x and $x^{'}$ are matched feature points. We define the error function as:

$$E = x^{'T} F x = x^{'T} l = x^{'T} [e]_{\times} \pi(x). \tag{4.1}$$

where F is the fundamental matrix [58], e the epipole of current frame, l the epipolar line, $[*]_{\times}$ the skew matrix, $\pi(x)$ projects x from the previous frame to current frame. The most import thing is to calculate the epipolar line. We detect outliers by calculating the distance between the projected feature points and their epipolar line.

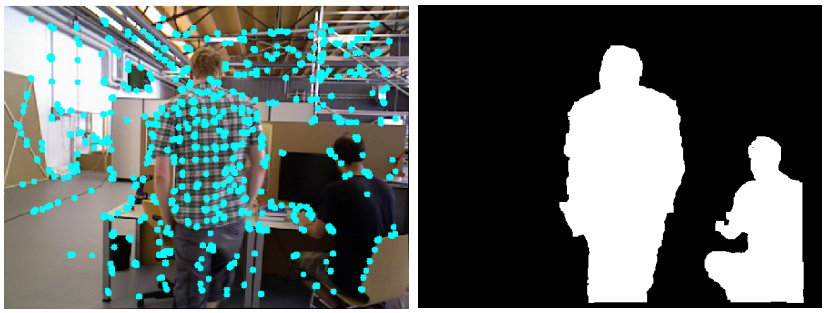


(a) RGB (Frame 680) (b) RGB (Frame 681)

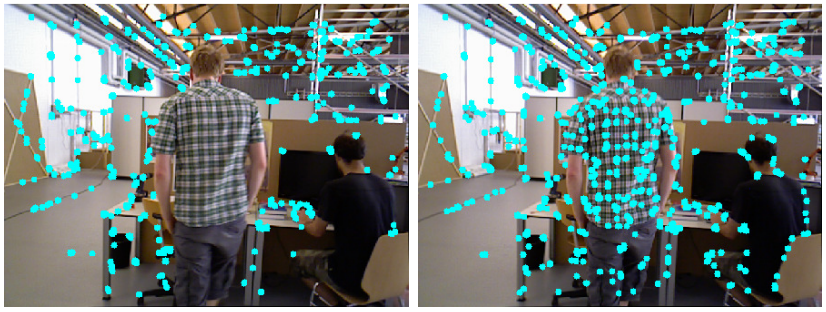
(c) depth (Frame 680) (d) depth (Frame 681)

(e) label (Frame 680) (f) label (Frame 681)

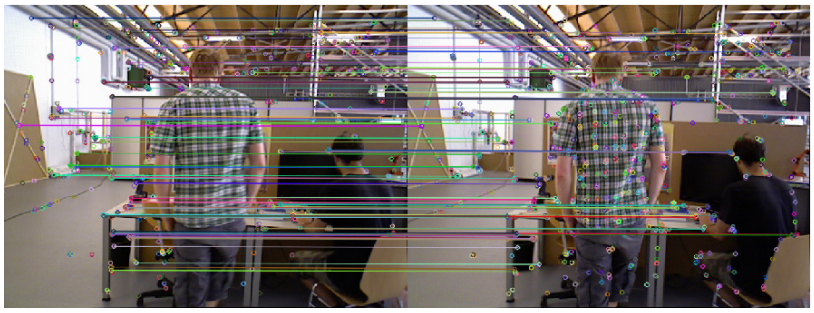
Figure 4-3: Segmentation result using the depth image. (b) is the current frame and (a) the previous frame. (e) and (f) are the segmentation results using K-means with 10 clusters.



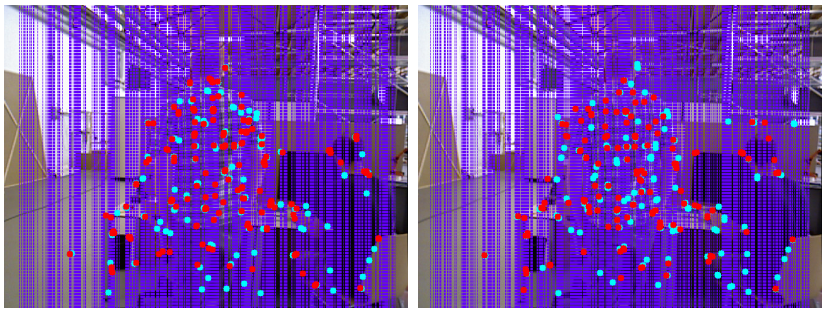
(a) Frame 680 (b) Frame 680



(c) Frame 680 (d) Frame 681



(e) Match Frame 680 and 681



(f) Frame 680 (g) Frame 681

Figure 4-4: Calculating fundamental matrix and Epipolar Line. (a) is the features (F_{t-1}) in previous image, (b) the mask of dynamic object in previous image, (c) the features (F'_{t-1}) after removing outliers using mask, (d) the features (F_t) of current image and (e) is the matching result using good matches of (a) and (b). (f, g) show the epipolar lines and outliers. Outliers are marked in red color, blue dots the other features and the purple line are epipolar lines.

Algorithm 5 Geometric Constraint

Require: Features of previous image: F_{t-1}, $F_{t-1}^{'}$
Features in current image: F_t
Hyper-parameters: $\lambda = 4, \beta = 30, \epsilon = 5$
Ensure: Outliers probability: PG
1: $M = match(F_t, F_{t-1}^{'})$
2: $GM = selectGoodMatches(M, \lambda, \beta)$
3: $F = CalculateFundamentalMatrix(GM)$
4: $M = match(F_t, F_{t-1})$
5: $l^{'} = calculateEpipolarLine(M, F)$
6: $E = calculateGeometricError(l^{'})$
7: **for** $\forall e \in E$ **do**
8: **if** $e < \epsilon$ **then**
9: $featue \in inlier$
10: **else**
11: $feature \in outlier$
12: **end if**
13: **end for**
14: $PG = calculateOutlierProbability()$
15: **return** PG

Calculating the Fundamental Matrix

Before calculating the epipolar line, we need to calculate the Fundamental Matrix F using selected well-matched corresponding features GM, as shown in Alg. 5 (*Lines 1-3*). First, we obtain the features in previous frame where all possible dynamic objects are removed using its mask, as shown in Figs. 4-4 (a-c). Then, GM is selected by matching current frame 4-4 (d) and synthetic previous frame 4-4 (c). Finally, F is calculated using GM, as shown in Fig. 4-4 (e).

Selecting Good Matches

The relatively good matches are selected when calculating the fundamental matrix, as shown in Alg. 6. Firstly, the minimum and maximum distance of ORB descriptors of matched ORB features are calculated. λ is a threshold, used to control the range of distance and β is also a threshold, which is used to decide the minimum number of matches needed to calculate the fundamental matrix. The higher λ and β are, the more matched features are reserved.

Algorithm 6 Select Good Matches

Require: Matched feature set: M
Parameters: $\lambda = 3$ ($\lambda >= 1$), $\beta = 30$ ($\beta > 8$)
Ensure: Good matches: GM

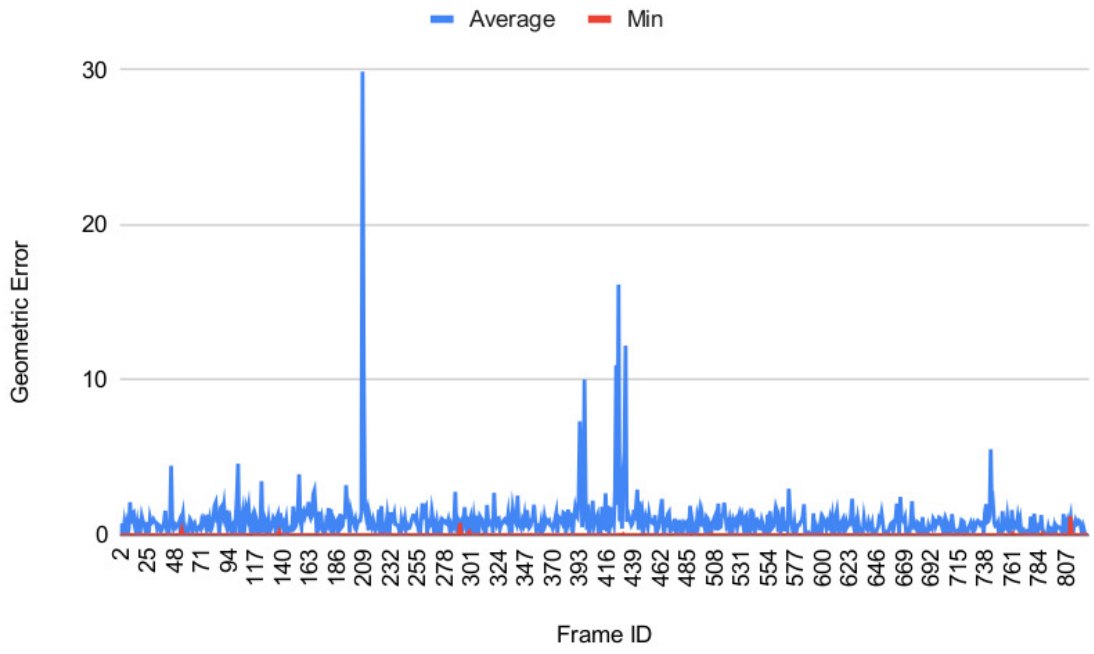
```
1: min, max = MinMaxDistance(M)
2: for λ; λ * min <= max; λ++ do
3:   for i = 0; i < M.size(); i++ do
4:     if M[i].distance < λ * min then
5:       GM.push_back(M[i])
6:     end if
7:     if GM.size() > β then
8:       break
9:     end if
10:  end for
11: end for
```

Calculating Epipolar Line

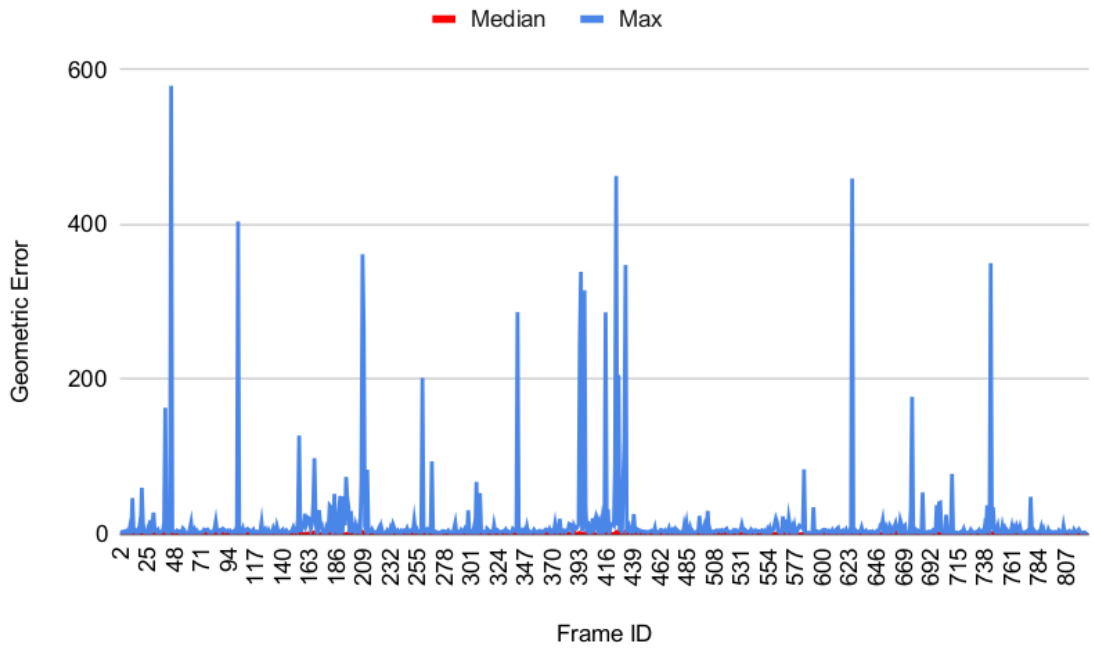
The epipolar line is calculated (Alg. 5 (*Lines 3-5*)) by matching original features in the previous and current frames. The results of epipolar lines are shown in Fig. 4-4 (g) using the matched features. In order to calculate the mask image of dynamic objects in the current frame, we need to match again using all the features, because only part of features are used in the matching operation of the previous step (Fig. 4-4 (e)). Fig. 4-4(f) is calculated using its previous frame (*Frame 679*).

Outliers Detection

Each features in the current image will be judged as an outliers only if the error E (Eq. 4.1) is larger than the threshold ϵ, which is decided using statistics, as shown in Fig. 4-5 and Alg. 5 (*Lines 6-13*). The maximum error is sometimes larger than 200 according to Fig. 4-5 (b), and the average value is below 5 for most frames, as shown in Fig. 4-5 (a). Therefore, ϵ is set to 5 in our experiment.

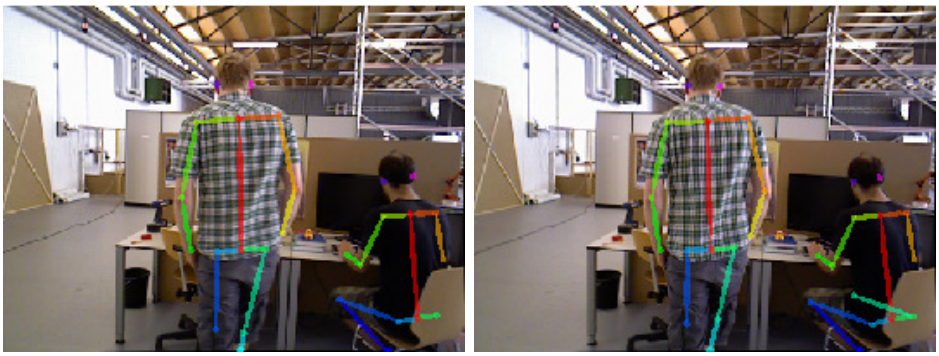


(a) Average vs minimum



(b) Median vs maximum

Figure 4-5: Geometric error distribution of fr3/walk_xyz dataset



(a) Frame 680 (b) Frame 681

Figure 4-6: Person detection

4.2.4 Moving Object Detection

Geometric Probability

The moving probability of each object using the geometric constraint is defined as:

$$PG_i = \frac{OUT_i}{OUT_i + IN_i} \tag{4.2}$$

where i $(0 \leq i < N)$ is the object segmented by K-means and N is the total number of objects; PG_i is the moving probability of each object based on the geometric constraint; OUT_i is the number of outliers in each object, and IN_i is the number of inlier features in each object.

OpenPose Probability

OpenPose [1] is used to detect the person, as shown in Fig. 4-6. The features of OpenPose are very sparse and very few features are detected for one person. However, these sparse features can locate the person using pixel value of the image. The moving probability using human detection is defined as:

$$PH_i = min\Big\{\frac{H_i}{H}, 1\Big\} \tag{4.3}$$

[1] https://github.com/firephinx/openpose_ros

Algorithm 7 Moving Object Detection

Require: Geometric probability: $PG[N]$
OpenPose probability: $PH[N]$
Segmentation result: $C_i, (0 < i < N)$
Thresholds: $\omega = 0.8, \theta = 0.25, \gamma = 0.5$
Ensure: Mask image of dynamic objects

```
for ∀i ∈ N do
    if PH[i] > θ, (0 < θ ≤ 1) then
        Person exists
    else
        No person exists
    end if
    P_i = Moving_object_probability(PG[i], PH[i], ω)
    if P_i > γ, (0 < γ ≤ 1) then
        Object C_i ∈ dynamic object
        Mask+ = C_i
    end if
end for
return Mask
```

where PH_i is the moving probability of each object based on human detection, H_i the number of OpenPose features on each object, and H a constant value, total OpenPose features on one person, which is set to 25 in the experiment.

Movable Object Probability

Moving objects are detected using both OpenPose and Geometric Constraint using:

$$P_i = \begin{cases} \omega PH_i + (1-\omega)PG_i, & \text{if person exists} \\ PG_i, & \text{otherwise} \end{cases} \tag{4.4}$$

where P_i is the moving probability of each object, and ω $(0 \leq \omega \leq 1)$ is a weight parameter indicating how much the result of human detection is trusted and is set to 0.8 in the experiment because the result of OpenPose is more trusted than that of Geometric Constraint.

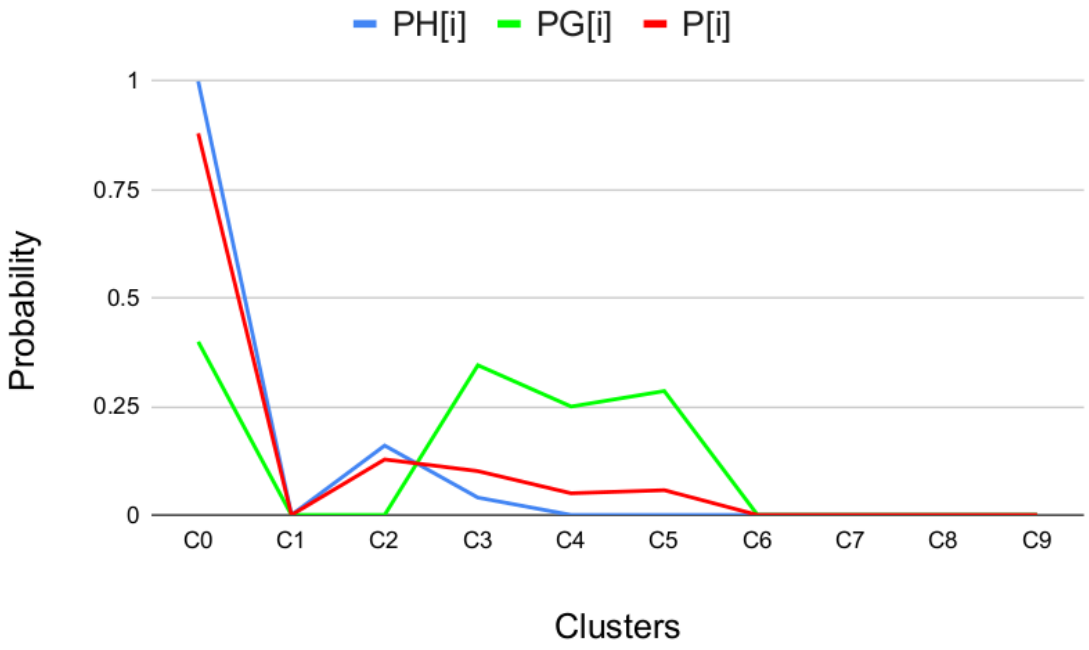
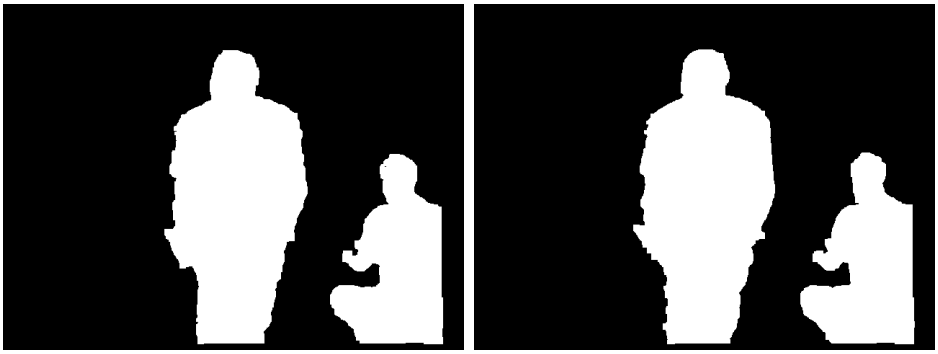


Figure 4-7: Moving object probability of Frame 681. $C_0 - C_9$ is the ten objects segmented by K-means.



(a) Frame 680 (b) Frame 681

Figure 4-8: Mask for removing dynamic objects

Judging Moving Object

The moving object detection algorithm is shown in Alg. 7, which generates the mask image of dynamic objects, which is used to remove outliers from the current frame. For each object, we firstly judge whether a person exists using the moving probability of PH_i of each frame, as shown in Fig. 4-7 (*blue line*). Geometric Constraint probability is shown in Fig. 4-7 (*green line*), which is inaccurate sometimes, but it can select the

Table 4.1: ATE against state-of-the-art RGB-D SLAM systems in dynamic scenes

	ORB SLAM2 (m)				New RGB-D SLAM [57] (m)				KMOP-vSLAM (m)			
	RMSE	Mean	Min	Max	RMSE	Mean	Min	Max	RMSE	Mean	Min	Max
fr3/walk_xyz	0.89	0.757	0.131	2.174	0.3047	0.2768	0.043	0.8196	**0.019**	**0.015**	**0.001**	**0.104**
fr3/walk_rpy	0.770	0.683	0.130	1.555	0.4983	0.4637	0.0708	0.9351	**0.049**	**0.039**	**0.004**	**0.253**
fr3/walk_half	0.366	0.337	0.024	0.965	0.3116	0.2972	0.1088	0.6246	**0.176**	**0.166**	**0.049**	**0.277**
fr3/walk_static	0.34	0.311	0.024	0.58	0.3080	0.2691	0.0231	0.6161	**0.032**	**0.023**	**0.001**	**0.205**

Table 4.2: Comparison of the RMSE of the RPE against state-of-the-art SLAMs in the dynamic scene

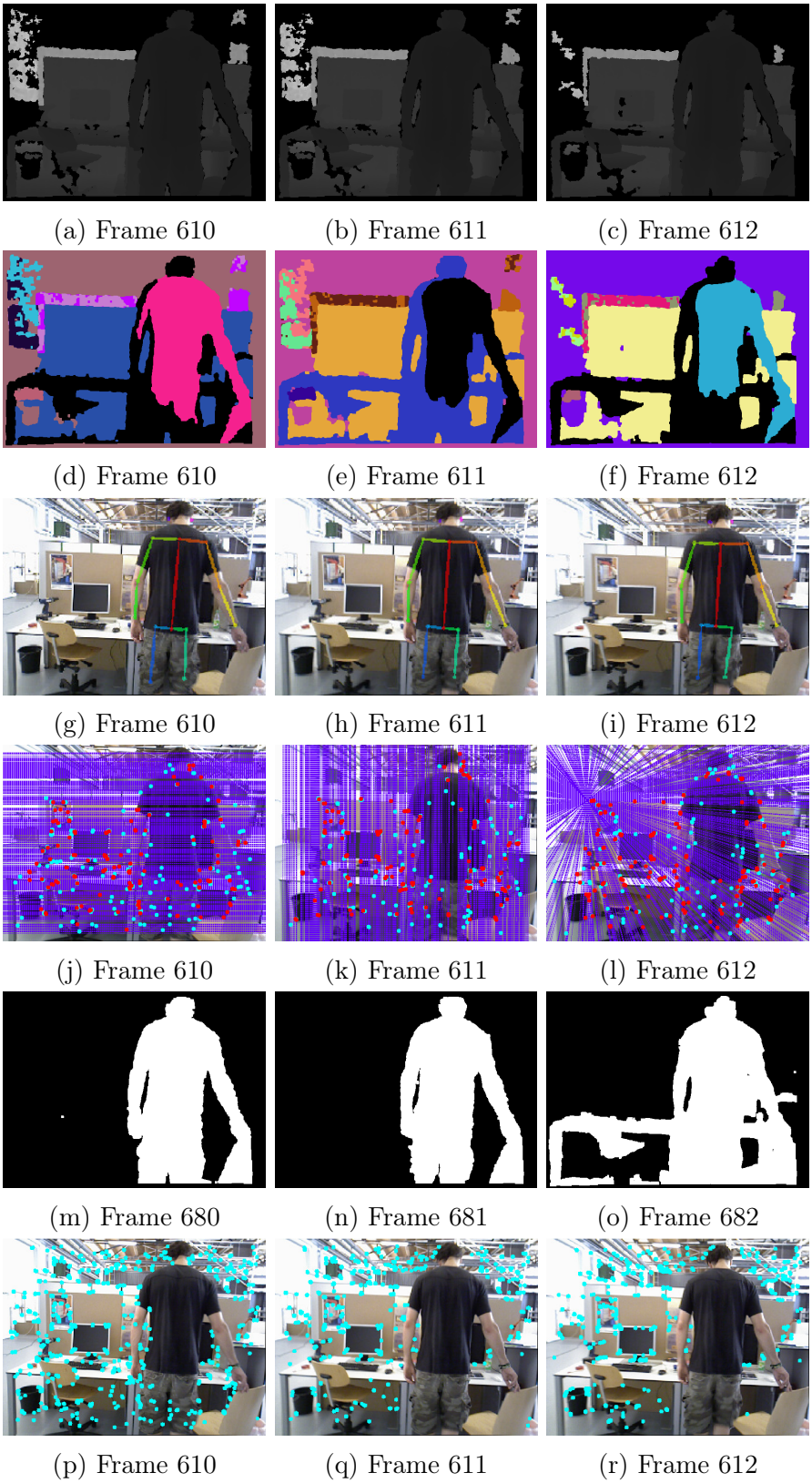
	Translation (m)				Rotation (deg)			
	ORB SLAM2	DVO	New RGB-D SLAM [57]	Ours	ORB SLAM2	DVO	New RGB-D SLAM [57]	Ours
fr3/walk_xyz	0.363	0.436	0.2158	**0.026**	6.698	7.6669	3.6476	**0.689**
fr3/walk_rpy	0.361	0.4038	0.3270	**0.065**	7.070	7.0662	6.3398	**1.105**
fr3/walk_half	0.28	0.2628	0.1908	**0.07**	5.469	5.2179	4.2863	**1.595**
fr3/walk_static	0.204	0.3818	0.1881	**0.033**	3.645	6.3502	3.2101	**0.627**

moving objects candidates. The moving probability P_i is calculated from PH_i and PG_i using Eq. (4.4). Finally, the object is judged as a dynamic object only if P_i is larger than the threshold γ. The mask image generated using these objects as shown in Fig. 4-8 and is used to remove outliers in tracking.

4.3 Experimental Results

4.3.1 TUM Dataset Evaluation

The TUM RGB-D dataset [49] contains indoor sequences from RGB-D sensors grouped in several categories to evaluate SLAM and odometry methods under different texture, illumination and structural conditions. There are sequences called *fr3/walk_ ** specially for evaluating SLAM in dynamic environments. In these sequences, two people walk through an office and there are four types of camera motions: halfsphere (half), xyz, rpy and static. The proposed system was evaluated using ATE and RPE metrics and compared with state-of-the-art SLAM systems using when possible the results published in the original papers. Tab. 4.1 shows the comparison of ATE error metric and Tab. 4.2 shows the comparison using the (Root Mean Square Error) RMSE of the RPE metric. The proposal outperformed [57], which is a similar work to this paper, in tracking accuracy.



(a) Frame 610 (b) Frame 611 (c) Frame 612

(d) Frame 610 (e) Frame 611 (f) Frame 612

(g) Frame 610 (h) Frame 611 (i) Frame 612

(j) Frame 610 (k) Frame 611 (l) Frame 612

(m) Frame 680 (n) Frame 681 (o) Frame 682

(p) Frame 610 (q) Frame 611 (r) Frame 612

Figure 4-9: Moving object detection result using f3/walk_xyz dataset. (a-c) depth image; (d-h) segmentation result; (g-i) OpenPose results; (j-l) geometric moving check; (m-o) mask of dynamic objects; (p-r) features after reducing outliers.

Table 4.3: Evaluation of time taken (fr3/walk_xyz)

Module	Person Detection	Segmentation	Tracking	Total
Time (ms)	45.89	95.29	221.636	257.819

4.3.2 Moving Object Detection

In the TUM dataset, the people and the chair someone is moving are dynamic objects, as shown in Fig. 4-9. In Fig. 4-9(m-r), the mask image of dynamic objects are generated successfully using K-means, human detection and multi-view geometric check and they can be used to improve the tracking accuracy of camera ego-motion.

4.3.3 Calculation Time

Intel i7 CPU and GeForce GTX 1080 Ti GPU are used in the experiment. GPU is used only for Human Detection. The time taken is shown in Tab. 4.3. *Total* is the average time cost for each frame and is similar to the time cost of *Tracking*, because *Tracking*, *Segmentation* and *Person Detection* are run in parallel.

4.4 Conclusions

In order to reduce the drift error in tracking caused by dynamic objects, including unknown objects, a novel approach is presented that combines geometric constraints, unsupervised learning segmentation, and human detection to detect moving objects and remove outliers. An efficient geometric outliers detection algorithm is also presented that utilizes the mask of dynamic objects of previous frames. In addition, a novel probability model that using geometric moving checks and human detection is proposed to judge dynamic objects. In the experiments, we evaluated our system using the TUM dataset under dynamic environments and the drift error of ATE and RPE are greatly reduced using the mask of dynamic objects. This work achieved good tracking performance. One problem is that the cluster number of K-means is given manually. But, K-means++ [59] proposed an algorithm for choosing the initial values for K-means algorithm. We will consider K-means++ in our future work to

apply the proposed method to cluttered environments.

Chapter 5

RDS-SLAM

Many solutions are proposed that use different kinds of semantic segmentation methods to detect dynamic objects and remove outliers. However, as far as we know, such kind of methods wait for the semantic results in the tracking thread in their architecture, and the processing time depends on the segmentation methods used. In this chapter, we present RDS-SLAM [13], a real-time visual dynamic SLAM algorithm that is built on ORB-SLAM3 and adds a semantic thread and a semantic-based optimization thread for robust tracking and mapping in dynamic environments in real-time. These novel threads run in parallel with the others, and therefore the tracking thread does not need to wait for the semantic information any more. Besides, we propose an algorithm to obtain as the latest semantic information as possible, thereby making it possible to use segmentation methods with different speeds in a uniform way. We update and propagate semantic information using the moving probability, which is saved in the map and used to remove outliers from tracking using a data association algorithm. Finally, we evaluate the tracking accuracy and real-time performance using the public TUM RGB-D datasets and Kinect camera in dynamic indoor scenarios.

5.1 Non-blocked Model and Contributions

Dynamic objects will cause many bad or unstable data associations that accumulate drifts during the SLAM process. In Fig. 5-1, for example, assume m_1 is on a person

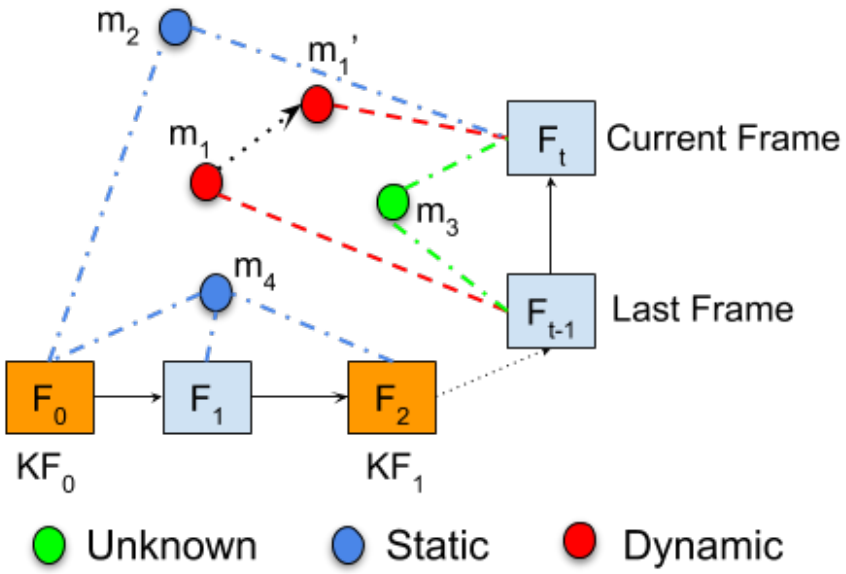
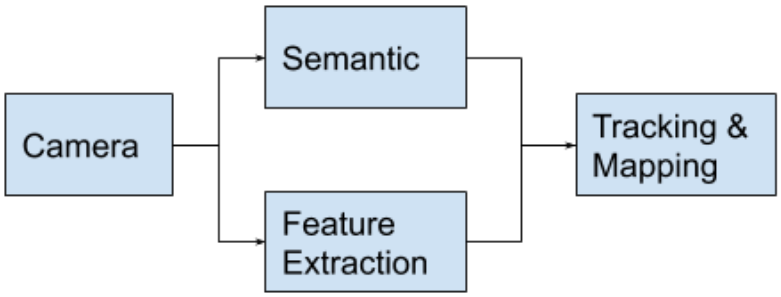


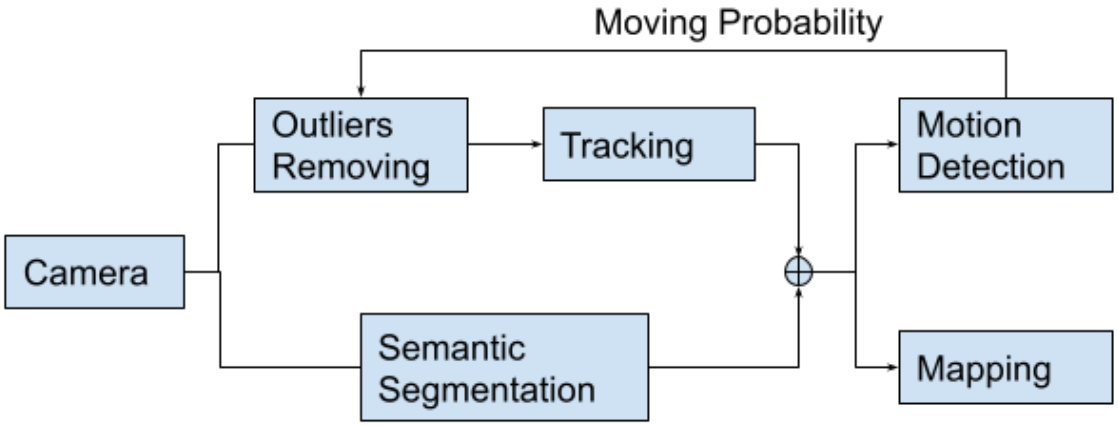
Figure 5-1: Example of data association in vSLAM under dynamic scene. $F_t, (t \geq 0)$ is the frame and KF_t is the selected keyframe. m_i, $i \in \{0, 1, ...\}$ is the map point. Assume m_1 moved to new position m_1' because it belongs to a moving object. The red line indicates the unstable or bad data association.

and its position changes in the scene. The bad or unstable data associations (the red lines in Fig. 5-1) will lead to incorrect camera ego-motion estimation in dynamic environments. Usually, there are two basic requirements for vSLAM: robustness in tracking and real-time performance. Therefore, how to detect dynamic objects in the populated scene and prevent the tracking algorithm from using data associations related to such dynamic objects in real-time is the challenge to allow vSLAM to be deployed in the real world.

We classify the solutions into two classes: pure geometric-based [29, 32–34, 56] and semantic-based [41, 43, 44, 46, 47, 60] methods. These geometric-based approaches cannot remove all potential dynamic objects, e.g., people who are sitting. Features on such objects are unreliable and also need to be removed from tracking and mapping. These semantic-based methods use semantic segmentation or object detection approaches to obtain pixel-wise masks or bounding box of potential dynamic objects. Sitting people can be detected and removed from tracking and mapping using the semantic information and a map of static objects can be built. Usually, in semantic-



(a) Blocked model.



(b) Non-blocked model.

Figure 5-2: Blocked model vs non-blocked model. The semantic model can use different kinds of segmentation methods, e.g., Mask R-CNN and SegNet. Note that this is not exactly the same as the semantic-based methods mentioned [41, 43, 44, 46, 47, 60]. In the blocked model, the tracking process is blocked to wait for the results of the semantic model.

based methods, geometric check, such as Random Sample Consensus (RANSAC) [7] and multi-view geometry, are also used to remove outliers.

These semantic-based methods first detect or segment objects and then remove outliers from tracking. The tracking thread has to wait for semantic information before tracking (camera ego-motion estimation), which is called the *blocked model* (as shown in Fig. 5-2 (a)). Their processing speed is limited by the time-consuming of semantic segmentation methods used. For example, Mask R-CNN requires about 200ms [24] for segmenting one image and this will limit the real-time performance of the entire system.

Our main challenge is how to execute vSLAM in real-time under dynamic scenes

with various pixel-wise semantic segmentation methods that ran at a different speed, such as SegNet and Mask R-CNN. We propose a semantic thread to wait for the semantic information. It runs in parallel with the tracking thread and the tracking thread does not need to wait for the segment result. Therefore, the tracking thread can execute in real-time. We call it a *non-blocked model*, as shown in Fig. 5-2 (b). Faster segmentation methods (e.g., SegNet) can update semantic information more frequently than slower methods (e.g., Mask R-CNN). Although we cannot control the segmentation speed, we can use a strategy to obtain as the latest semantic information as possible to remove outliers from the current frame.

Because the semantic thread runs in parallel with the tracking thread, we use the map points to save and share the semantic information. As shown in Fig. 5-1, we update and propagate semantic information using the moving probability and classify map points into three categories, static, dynamic, and unknown, according to the moving probability threshold. These classified map points will be used to select as stable data associations as possible in tracking.

The main contributions of RDS-SLAM are:

(1) we propose a novel semantic-based real-time dynamic vSLAM algorithm, RDS-SLAM, which enables the tracking thread does not need to wait for the semantic results anymore. This method efficiently and effectively uses semantic segmentation results for dynamic object detection and outlier removal while keeping the algorithm's real-time nature.

(2) we propose a keyframe selection strategy that uses as the latest new semantic information as possible for outliers removal with any semantic segmentation methods with different speeds in a uniform way.

(3) We show the real-time performance of the proposed method is better than the existing similar methods using the TUM dataset.

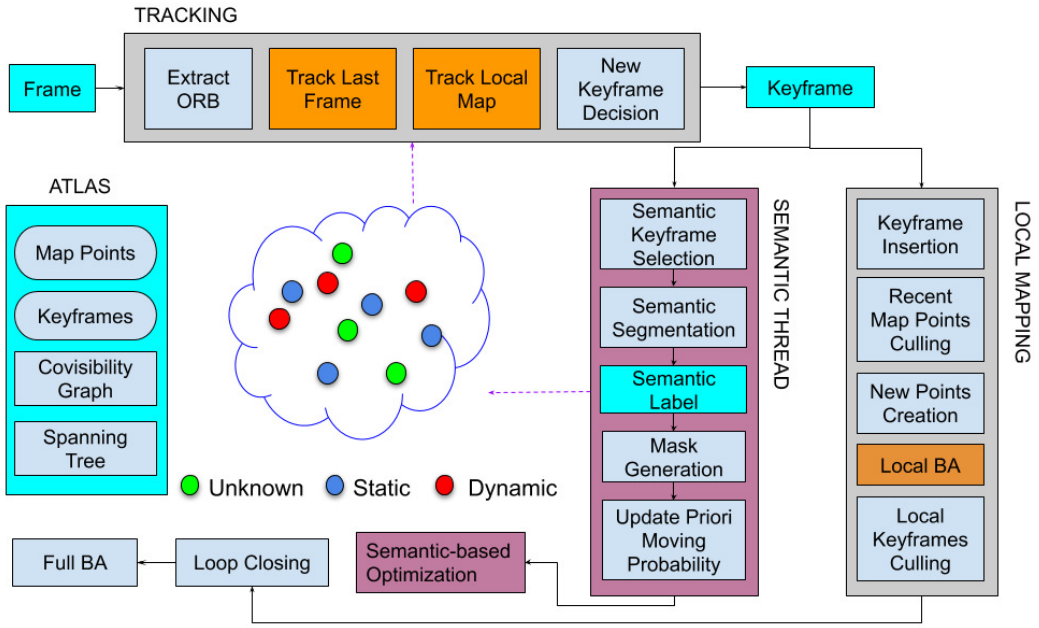


Figure 5-3: System architecture. Models with orange color are modified blocks based on ORB-SLAM3. Models with magenta color are newly added features. Blocks in blue are important data structures.

5.2 System Overview

Each frame will first pass through the tracking thread. The initial camera pose is estimated for the current frame after being tracked with the last frame and further optimized by being tracked with the local map. Then, keyframes are selected and they are useful in semantic tracking, semantic-based optimization, and local mapping thread. We modify several models in the tracking and the local mapping threads to remove outliers from camera ego-motion estimation using the semantic information. In the tracking thread, we propose a data association algorithm to use as the features on static objects as possible.

The semantic thread runs in parallel with the others, so as not to block the tracking thread and saves the semantic information into the atlas. Semantic labels are used to generate the mask image of the priori dynamic objects. The moving probability of the map points matched with features in the keyframes is updated using the semantic information. Finally, the camera pose is optimized using the semantic information in the atlas.

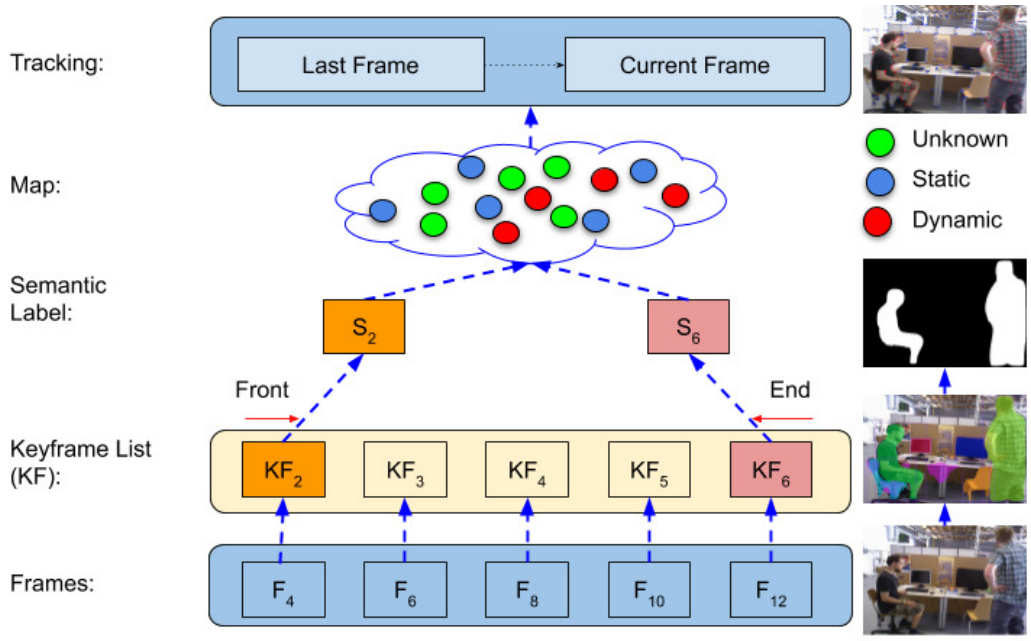


Figure 5-4: Semantic tracking example. Assume keyframes KF_n is selected every two frames F_n and inserted into keyframe list KF. We choose keyframes from KF to request semantic labels S_n. Then we update the moving probability into the atlas using the mask image of dynamic objects that reproduced from the semantic label. Blue circles stand for the static map points and red circles for the dynamic map points. Others marked in green are unknown.

We will introduce the new features and modified models in the following sections. We skip the detailed explanations of the modules that are the same as those of ORB-SLAM3.

5.3 Semantic Thread

The semantic thread is responsible for generating semantic information and updating it into the atlas map. Before we introduce the detailed implementation of the semantic thread, we use a simple example to explain the general flow, as shown in Fig. 5-4. Assume the keyframes are selected every two frames. The keyframes are selected by the ORB-SLAM3 and we inserted them into a keyframe list KF sequentially. Assume, at time $t = 12$, KF_2-KF_6 are inside KF. The next step is to select keyframes from KF to request semantic labels from the semantic server. We call this process as *semantic keyframe selection* process. We take one keyframe from the head of KF

Algorithm 8 Semantic Tracking Thread

Require: KeyFrame list: KF

```
1: while not_request_finish() do
2:    SK = semantic_keyframe_selection(KF)
3:    SLs = request_segmentation(SK)
4:    for i = 0; i < SLs.size(); i++ do
5:       KeyFrame kf = SR[i]
6:       kf->mask = GenerateMaskImage(SLs[i])
7:       kf->UpdatePrioriMovingProbability()
8:    end for
9: end while
```

(KF_2) and one from the back of KF (KF_6) to request the semantic labels. Then, we calculate the mask of the priori dynamic objects using semantic labels S_2 and S_6. Next, we update the moving probability of map points stored in the atlas. The moving probability will be used later to remove outliers from the tracking thread.

Alg. 8 shows the detailed implementation of the semantic thread. The first step is to select semantic keyframes from keyframe list KF (Line 2). Next, we request semantic labels from the semantic model and return the semantic labels SLs (Line 3). Lines 4-8 are to save and process the semantic results for each item returned. Line 6 is to generate the mask image of dynamic objects and Line 7 updates the moving probability stored in the atlas. We will introduce each submodule of the semantic thread sequentially (see Fig. 5-3).

5.3.1 Semantic Keyframe Selection Algorithm

The semantic keyframe selection algorithm is to select keyframes for requesting the semantic labels later. We need to keep the real-time performance while using different kinds of semantic segmentation methods. However, some of them, such as Mask R-CNN, are time-consuming and the current frame in tracking may not obtain the new semantic information if we segment every keyframe sequentially.

To evaluate the distance quantitatively, we define the *semantic delay* that is the distance between the latest frame id which has the semantic label (S_t) that holds the

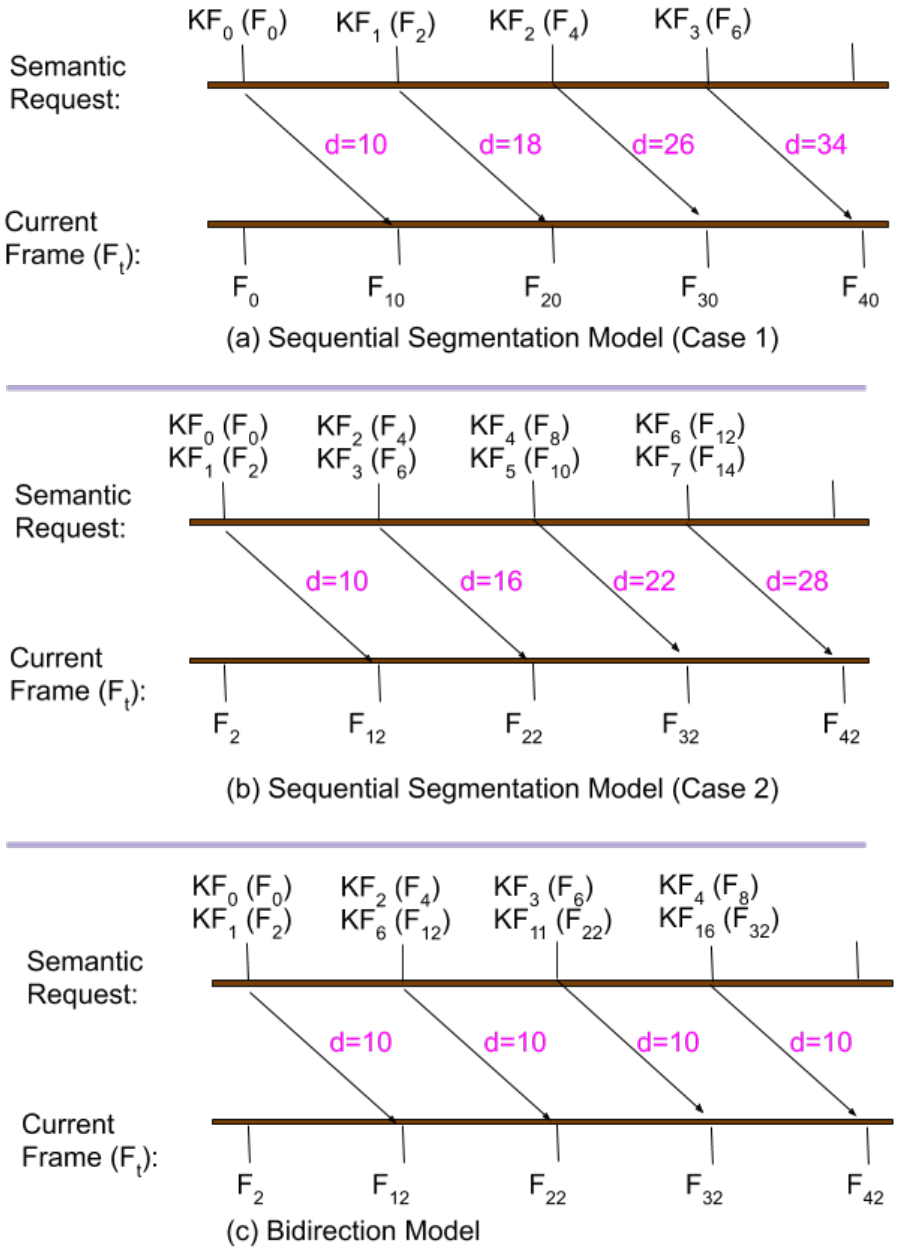


Figure 5-5: Bi-direction model vs sequential model. Assume we use Mask R-CNN (200ms) and ORB-SLAM3 (20ms), and the keyframe is selected every two frames. About $200/20 = 10$ frames delay while waiting for the semantic result.

latest semantic information and the current frame (F_t) id, as follows:

$$d = FrameID(F_t) - FrameID(S_t). \tag{5.1}$$

Fig. 5-5 shows the semantic delay for several cases. The general idea is to segment

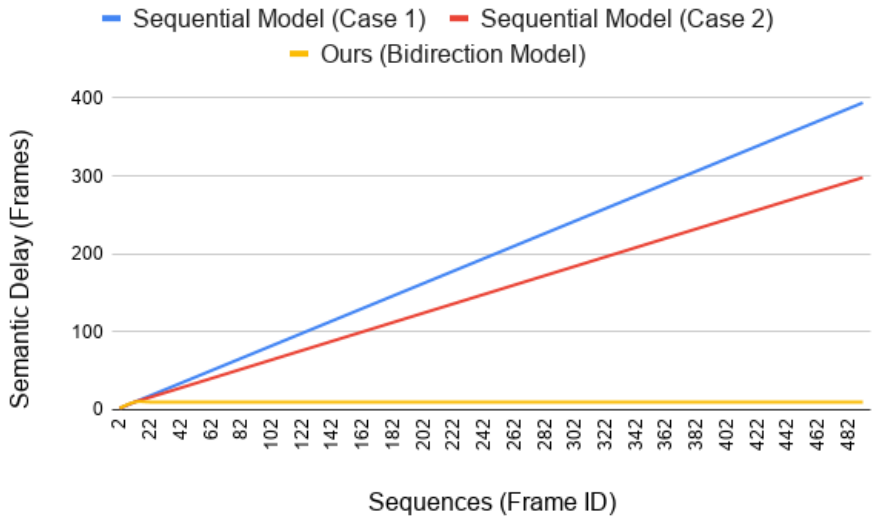


Figure 5-6: Semantic delay of sequential model vs bi-direction model.

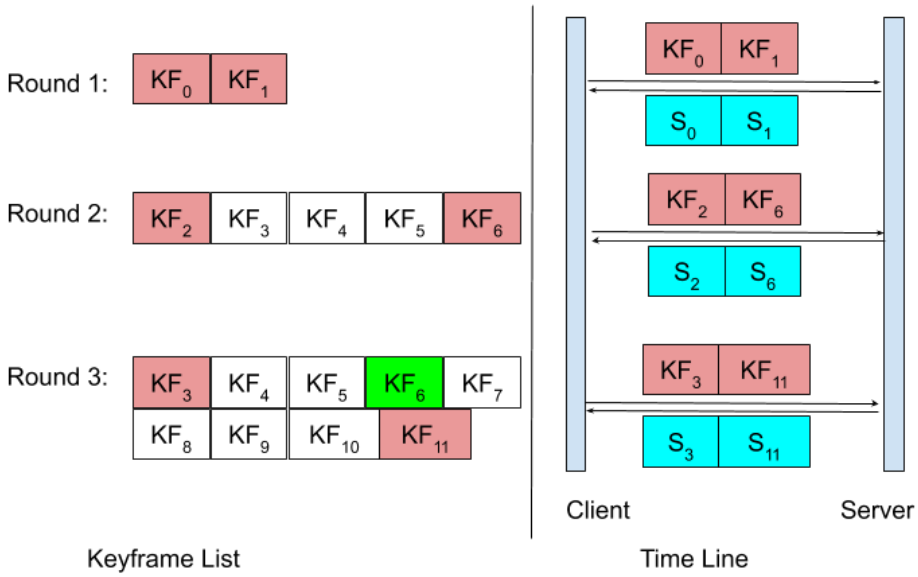


Figure 5-7: Semantic time line. The left side is the contents inside the keyframe list KF and right side is the time line of requesting semantic label. Keyframe in green color means this item has already obtained the semantic information in the previous round.

each frame or keyframe sequentially, according to the time sequence as shown in Fig. 5-5 (a). We call this kind of model the *sequential segmentation model*. However, this will monotonically increase the time delay when using time-consuming segmentation methods as shown as the blue line in Fig. 5-6. For instance, at time $t = 10$ (F_{10}),

the semantic model completed the segmentation of KF_0 (F_0) and the semantic delay is $d = 10$. Similarly, at time 40 (F_{40}), the semantic delay becomes 34. That is, the last frame that has semantic information is 34 frames behind the current frame. The current frame cannot obtain the latest semantic information.

To shorten the distance, supposed that we segment two frames sequentially at the same time (Fig. 5-5 (b)). Then, the delay becomes $12 - 2 = 10$ if KF_0 and KF_1 are segmented at the same time. The delay still grows linearly as shown as the red line in Fig. 5-6.

To further shorten the semantic delay, we use a *bi-directional model*. We do not segment keyframes sequentially. Instead, we do semantic segmentation using keyframes both from the front and back of the list to use as the latest semantic information as possible, as shown in Fig. 5-5 (c) and as the yellow line in Fig. 5-6. The semantic delay becomes a constant value. In practice, the delay in the bidirectional model is not always 10. The distance is influenced by the segmentation method used, the frequency of keyframe selection, and the processing speed of the related threads.

The left side of Fig. 5-7 indicates a semantic keyframe selection example and the right side of Fig. 5-7 shows the timeline of requesting semantic information from the semantic model/server. We take both keyframes from the head and back of KF to request the semantic label. (Round 1) At time $t = 2$, two keyframes KF_0 and KF_1 are selected. Segmentation finished at $t = 12$. By this time, new keyframes are selected and then inserted into KF (see Round 2). Then we take two elements KF_2 from the front and KF_6 from this back to request the semantic label. At the time $t = 22$, we received the semantic result and continue the next round (Round 3).

We can obtain relatively new information if we segment the keyframe at the tail of the KF list. Then why do we also need to segment the keyframe that in front of the list? Different from the blocked model, there is no semantic information for the first few frames (about 10 frames if using Mask R-CNN) in our method. Since the processing speed of the tracking thread is usually faster than the semantic thread, vSLAM may have already accumulated large errors because of the dynamic objects.

Therefore, we need to correct these drift errors using the semantic information by popping out and feeding the keyframes in the front of the KF list sequentially to the semantic-based optimization thread to correct/optimize the camera poses.

5.3.2 Semantic Segmentation

In our experiment, we use two models with different speeds, Mask R-CNN (slower) and SegNet (faster), as shown in Fig. 5-8. Mask R-CNN [24] is trained with the MS COCO [61], which has both pixel-wise semantic segmentation results and instance labels. We implemented it based on the TensorFlow version of Matterport [1]. SegNet [22] implemented using Caffe [2], is trained with the PASCAL VOC [50] 2012 dataset, where 20 classes are offered. We did not refine the network using the TUM dataset because SLAM usually runs in an unknown environment.

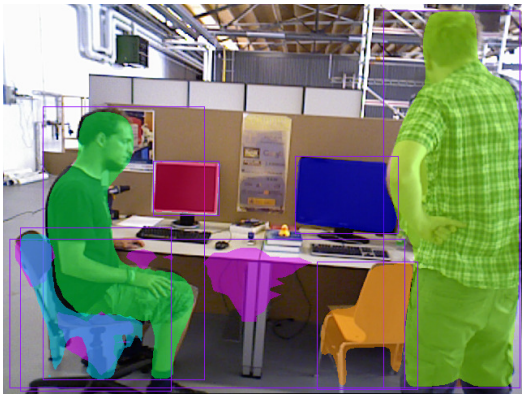
5.3.3 Semantic Mask Generation

We merge all the binary mask images of instance segmentation results into one mask image that is used to generate the mask image (Fig. 5-8) of people. Then we calculate the priori moving probability of map points using the mask. In practice, since the segmentation on object boundaries are sometimes unreliable, the features on the boundaries cannot be detected if directly apply the mask image, as shown in Fig. 5-9 (a). Therefore, we dilate the mask using a morphological filter to include the edge of dynamic objects, as shown in Fig. 5-9 (b).

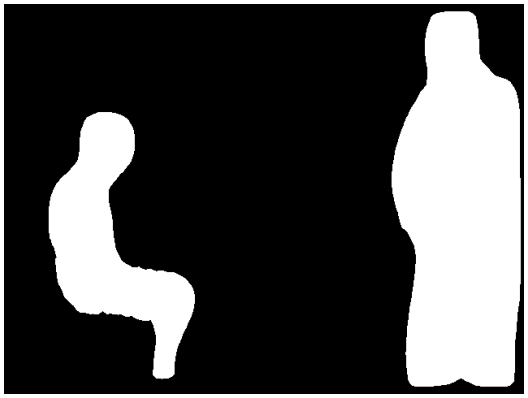
5.3.4 Moving Probability Update

In order not to wait for the semantic information in the tracking thread, we isolate the semantic segmentation from tracking. We use the moving probability to convey semantic information from semantic thread to tracking thread. The moving probability is used to detect and remove outliers from tracking.

[1] `https://github.com/matterport/Mask_RCNN`
[2] `https://github.com/alexgkendall/SegNet-Tutorial`



(a) (M) Segmentation result



(b) (M) Mask image

(c) (S) Segmentation result

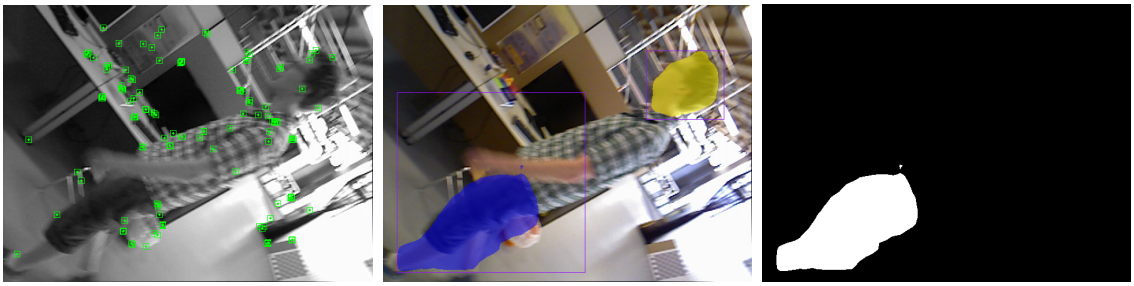
(d) (S) Mask image

(e) Detected outliers using mask image

Figure 5-8: Semantic information. "M" stands for Mask R-CNN and "S" for "SegNet". (e) shows the outliers that marked in red color, which are detected using the mask image.

(a) Original (b) Enlarged

Figure 5-9: Mask dilation. Remove outliers on the edge of dynamic objects.



(a) Features (b) Segmentation (c) Mask of Person

Figure 5-10: Segmentation failure case. Some features on the body on the person (a) cannot be identified as outliers using unsound mask (c) generated by semantic result (b). Therefore, those features are wrongly labeled as static in this frame.

Figure 5-11: Moving probability. θ_s is the static threshold and θ_d is the dynamic threshold value.

Definition of Moving Probability

As we know, vSLAM is usually running in an unknown environment, the semantic result is not always robust if the CNN network is not well trained or refined according to the current environment (Fig. 5-10). To detect outliers, it is more reasonable to consider the spatio-temporal consistency of frames, rather than just use the semantic

result of one frame. Therefore, we use the moving probability to leverage the semantic information of successive keyframes.

We define the moving probability ($p(m_t^i), m_t^i \in M$) of each map point i at the current time as shown in Fig. 5-11. The status of the map point is more likely dynamic if its moving probability is closer to one. The more static the map point is if it is more closer to zero. To simplify, we abbreviate the moving probability of map point i at time t ($p(m_t^i)$) to $p(m_t)$. Each map point has two status (M), dynamic and static, and the initial probability (initial belief) is set to 0.5 ($bel(m_0)$).

$$M = \{static(s), dynamic(d)\},$$
$$bel(m_0 = d) = bel(m_0 = s) = 0.5.$$

Definition of Observed Moving Probability

Considering the fact that the semantic segmentation is not 100% accurate, we define the observe moving probability as:

$$p(z_t = d|m_t = d) = \alpha,$$
$$p(z_t = s|m_t = d) = 1 - \alpha,$$
$$p(z_t = s|m_t = s) = \beta, \ and$$
$$p(z_t = d|m_t = s) = 1 - \beta.$$

The values α and β are manually given and it is related to the accuracy of semantic segmentation. In the experiment, we set α and β to 0.9 by supping the semantic segmentation is fairly reliable.

Moving Probability Update

The moving probability of the current time $bel(m_t)$ is predicted based on the observation $z_{1:t}$ (semantic segmentation) and initial status m_0. We formulate the moving

probability updating problem as a Bayesian filter [62] problem:

$$
\begin{aligned}
bel(m_t) &= p(m_t|z_{1:t}, m_0) \\
&= \eta p(z_t|m_t, z_{1:t-1}, m_0) p(m_t|z_{1:t-1}, m_0) \\
&= \eta p(z_t|m_t) p(m_t|z_{1:t-1}, m_0) \\
&= \eta p(z_t|m_t) \overline{bel}(m_t).
\end{aligned} \tag{5.2}
$$

In Eq. 5.2 exploits Bayes rule and the conditional independence that the current observation z_t only relies on the current status m_t. η is a constant. The prediction $\overline{bel}(m_t)$ is calculated by:

$$
\begin{aligned}
&\overline{bel}(m_t) \\
&= \int p(m_t|m_{t-1}, z_{1:t-1}) p(m_{t-1}|z_{1:t-1}) dm_{t-1} \\
&= \int p(m_t|m_{t-1}) bel(m_{t-1}) dm_{t-1}.
\end{aligned} \tag{5.3}
$$

In Eq. (5.3), we exploit the assumption that our state is complete. This implies if we know the previous state m_{t-1}, past measurements convey no information regarding the state m_t. We assume the state transition probability $p(m_t = d|m_{t-1} = s) = 0$ and $p(m_t = d|m_{t-1} = d) = 1$ because we cannot detect the suddenly change of objects. η is calculated by $(bel(m_t = d) + bel(m_t = s))/2$. The probability of map points belonging to dynamic is calculated by:

$$
\begin{aligned}
&\overline{bel}(m_t = d) \\
&= p(m_t = d|m_{t-1} = d) bel(m_{t-1} = d).
\end{aligned} \tag{5.4}
$$

Judgement of Static and Dynamic Points

Whether a point is dynamic or static is judged using predefined probability thresholds, θ_d and θ_s (see Fig. 5-11). They are set to 0.6 and 0.4 respectively in the experiment.

$$Status(m_t^i) = \begin{cases} dynamic & p(m_t) > \theta_d \\ static & p(m_t) < \theta_s \\ unknown & others \end{cases} . \tag{5.5}$$

5.4 Tracking Thread

The tracking thread runs in real-time and tends to accumulate the drift error due to the incorrect or unstable data associations of 3D map points and 2D features in each frame caused by dynamic objects. We modify the Track Last Frame model and Track Local Map model of ORB-SLAM3 tracking thread to remove outliers (see Fig. 5-3). We propose a data association algorithm that uses as good data associations as possible using the moving probability stored in the atlas.

5.4.1 Track Last Frame

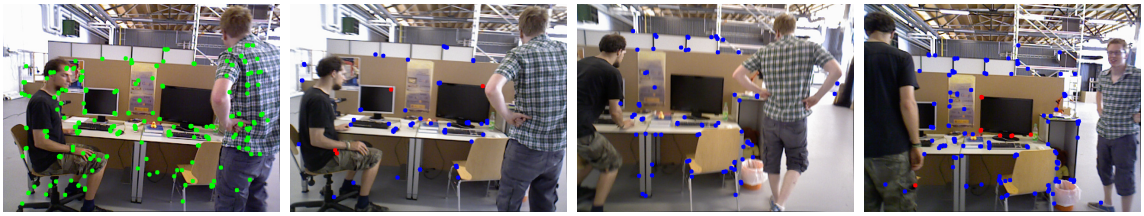
Alg. 9 shows the data association algorithm in tracking last frame model. For each feature i in the last frame, we first get their matched map point m (Line 2). Next, we find the matched feature in the current frame by comparing the descriptor distance of ORB features (Line 3). After that, in order to remove the bad influence from dynamic map points, we skip those map points that have higher moving probability (Lines 4-6). Then, there are two kinds of map points left, static and unknown map points. We want to use only the static map points as far as we can. Therefore, we classify the remaining map points into two subsets: static subset and unknown subset, according to their moving probability (Lines 7-12). Finally, we use the selected relative good matches. We first use all the good data stored in static subset (Lines 14-16). If the size of these data is not enough (less than the threshold $\tau = 20$, the value used in ORB-SLAM3), we also use the data in unknown subset (Lines 17-21).

Algorithm 9 Robust data association algorithm

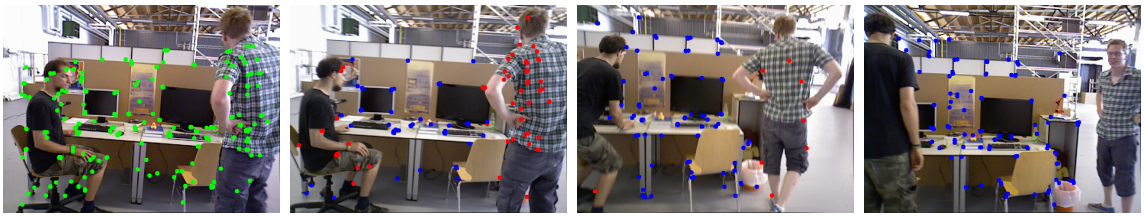
Require: Current Frame: F_t
Last Frame: F_{t-1}
Unknown subset: Unknown<FeatureId, MapPoint*>
Static subset: Static<FeatureId, MapPoint*>
Threshold: θ_d, θ_s, $\tau = 20$

```
for i=0; i<F_{t-1}.Features.size(); i++ do
    MapPoint* m = F_{t-1}.MapPoints[i]
    f = FindMatchedFeatures(F_t, m)
    if p(m) > θ_d then
        continue
    end if
    if p(m) < θ_s then
        Static.insert(f, m)
    end if
    if θ_d ≤ p(m) ≤ θ_s then
        Unknown.insert(f, m)
    end if
end for
for it = Static.begin(); it!=Static.end();it++ do
    F_t.MapPoints[it->first] = it->second;
end for
if Static.size()<τ then
    for it = Unknown.begin(); it!=Unknown.end();it++ do
        F_t.MapPoints[it->first] = it->second;
    end for
end if
```

We try to exclude outliers from tracking using the moving probability stored in the atlas. How well the outliers are removed will have a great influence on the tracking accuracy. We show the results of a few frames in Fig. 5-12. All the features in the first few frames are in green color because no semantic information can be used and the moving probability of all map points is 0.5, the initial value. The features in red belong to dynamic objects and they are hard to match with the last frame than static features (blue features). The green features are almost disappeared because the map points obtained the semantic information over time. We only use features in the static subset if its size number is enough to estimate camera ego-motion.

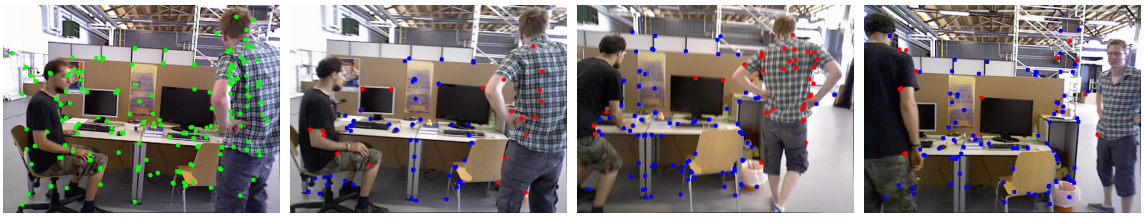


(a) (S) Frame 2 (b) (S) Frame 11 (c) (S) Frame 50 (d) (S) Frame 100

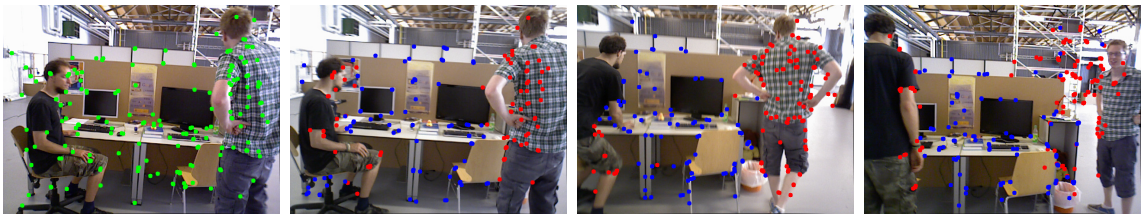


(e) (M) Frame 2 (f) (M) Frame 11 (g) (M) Frame 50 (h) (M) Frame 100

Figure 5-12: Results after tracking last frame. "M" stands for Mask R-CNN and "S" for SegNet. The features in red color are not used in tracking. Blue features belong to the static subset and green features belong to the unknown subset.



(a) (S) Frame 2 (b) (S) Frame 11 (c) (S) Frame 50 (d) (S) Frame 100



(e) (M) Frame 2 (f) (M) Frame 11 (g) (M) Frame 50 (h) (M) Frame 100

Figure 5-13: Results after tracking local map. "M" stands for Mask R-CNN and "S" for SegNet.

5.4.2 Track Local Map

The basic idea of the data association algorithm in the Tracking Local Map model is similar with Alg. 9. The difference is that here we use all the map points in the local map to find good data association. The data association result after tracking local map is shown in Fig. 5-13. More map points are used to match in this model than the tracking last frame. The features on the people are almost successfully detected

or not matched/used.

5.5 Optimization

5.5.1 Semantic-based Optimization

We optimize the camera pose using the keyframes given by the semantic keyframe selection algorithm. Considering that the tracking thread runs very fast than the semantic thread, drifts have already accumulated to some extent with the influence of dynamic objects. Therefore, we try to correct the camera pose using semantic information. We modify the error term used in ORB-SLAM3 by using the moving probability of map points for weighting, as shown below. In the experience, we only use the matched static map points for optimization.

Assume $X_j^w \in \mathbb{R}^3$ is the 3D pose of a map point j in the world coordinate system. The i-th keyframe pose in the world coordinate is $T_i^w \in SE(3)$. The camera pose T_i^w is optimized by minimizing the reprojection error concerning the matched keypoint $x_{ij} \in \mathbb{R}^2$ of the map point. The error term for the observation of a map point j in a keyframe i is:

$$e(i,j) = (x_{ij} - \pi_i(T_i^w, X_j^w))(1 - p(m^j)), \tag{5.6}$$

where π_i is the projection function that projects a 3D map point into a 2D pixel point in the keyframe i. The larger the moving probability is, the smaller contribution to the error. The cost function to be optimized is:

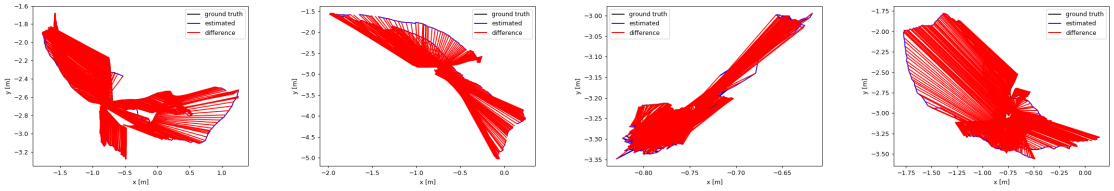
$$C = \sum_{i,j} \rho(e_{i,j}^T \Omega_{i,j}^{-1} e_{i,j}) \tag{5.7}$$

where ρ is the Huber robust cost function and $\Omega_{i,j}^{-1}$ is the covariance matrix.

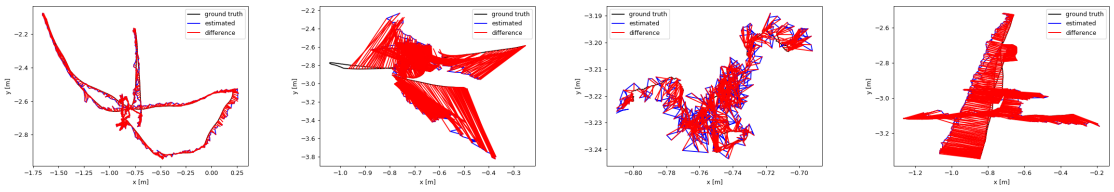
5.5.2 Bundle Adjustment in Local Mapping Thread

We modify the local BA model to reduce the influence of dynamic map points using semantic information. What we modified are: 1) the error term, in which the moving

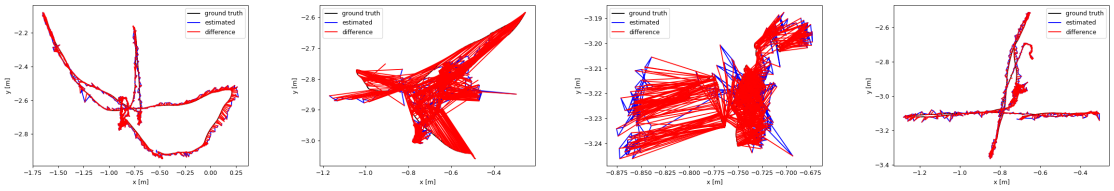
probability is used, as shown in Eq. 5.6; 2) only keyframes that already obtained semantic information are used for BA.



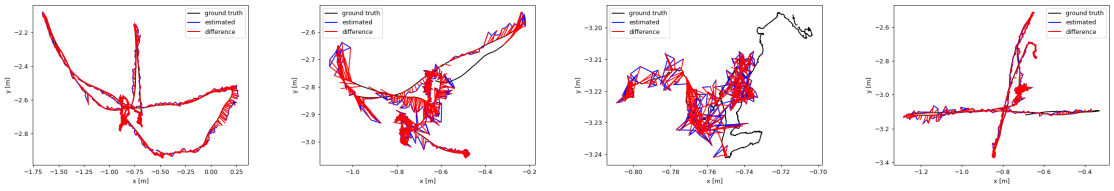
(a) ORB-SLAM3: w/half (b) ORB-SLAM3: w/rpy (c) ORB-SLAM3: w/static (d) ORB-SLAM3: w/xyz



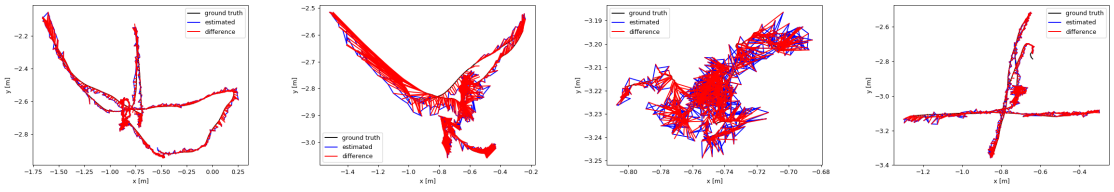
(e) DS-SLAM (S): w/half (f) DS-SLAM (S): w/rpy (g) DS-SLAM (S): w/static (h) DS-SLAM (S): w/xyz



(i) RDS-SLAM (S): w/half (j) RDS-SLAM (S): w/rpy (k) RDS-SLAM (S): w/static (l) RDS-SLAM (S): w/xyz



(m) DynaSLAM (M): w/half (n) DynaSLAM (M): w/rpy (o) DynaSLAM (M): w/static (p) DynaSLAM (M): w/xyz



(q) RDS-SLAM (M): w/half (r) RDS-SLAM (M): w/rpy (s) RDS-SLAM (M): w/static (t) RDS-SLAM (M): w/xyz

Figure 5-14: Trajectory comparing frame by brame. "M" stands for "Mask R-CNN" and "S" for "SegNet".

Table 5.1: Results of absolute trajectory error of TUM (m). RDS-SLAM (1) and (3) are evaluated results only using keyframes.

Seq.	ORB SLAM3		DS-SLAM		DynaSLAM)		SLAM-PCD		DM-SLAM		Detect-SLAM	RDS-SLAM (1) KeyFrame (Mask R-CNN)		RDS-SLAM (2) All (Mask R-CNN)		RDS-SLAM (3) KeyFrame (SegNet)		RDS-SLAM (4) All (SegNet)	
	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.
w/half	0.6572	0.3124	0.0303	0.0159	0.0296	0.0157	**0.0241**	0.0122	0.0274	0.0137	0.0514	0.0306	0.0171	0.0259	0.0141	0.0291	0.0143	0.0807	0.0454
w/rpy	1.0197	0.5122	0.4442	0.2350	0.0354	0.019	0.0453	0.0316	0.0328	0.0194	0.2959	0.0587	0.0380	0.1468	0.1051	**0.0128**	0.0081	0.1604	0.0873
w/static	0.3614	0.1522	0.0081	0.0033	**0.0068**	0.0032	0.0077	0.0039	0.0079	0.0040	-	0.0720	0.0343	0.0815	0.0224	0.0215	0.0104	0.0206	0.012
w/xyz	0.9178	0.4859	0.0247	0.0161	0.0164	0.0086	0.0157	0.0084	**0.0148**	0.0072	0.0241	0.0240	0.0139	0.0213	0.0127	0.0565	0.0184	0.0571	0.0229
s/static	0.0090	0.0043	0.0065	0.0033	0.0108	0.0056	0.0080	0.0037	**0.0063**	0.0032	-	0.0084	0.0043	0.0088	0.0043	0.0039	0.0017	0.0084	0.0043

Table 5.2: Results of translational relative pose error (RPE) (m). RDS-SLAM (1) and (3) are evaluated results only using keyframes.

Seq.	ORB SLAM3		DS-SLAM		DynaSLAM		SLAM-PCD		RDS-SLAM (1) KeyFrame (Mask R-CNN)		RDS-SLAM (2) All (Mask R-CNN)		RDS-SLAM (3) KeyFrame (SegNet)		RDS-SLAM (4) All (SegNet)	
	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.
w/half	0.3262	0.2625	0.0297	0.0152	0.0284	0.0149	0.0274	0.0140	0.0332	0.0208	0.0282	0.0155	**0.0274**	0.0140	0.0482	0.036
w/rpy	0.4368	0.3197	0.1503	0.1168	0.0448	0.0262	0.0616	0.0357	0.0700	0.0488	0.1114	0.0920	**0.0245**	0.0122	0.1320	0.1067
w/static	0.7800	0.7563	0.0102	0.0038	**0.0089**	0.0044	0.0102	0.0049	0.0529	0.0444	0.0419	0.0348	0.0221	0.0149	0.0221	0.0149
w/xyz	0.4258	0.3063	0.0333	0.0229	0.0217	0.0119	**0.0204**	0.0107	0.0299	0.4943	0.0281	0.0167	0.0269	0.0163	0.0426	0.0317
s/static	0.0102	0.0049	0.0078	0.0038	0.0126	0.0067	0.0087	0.0038	0.0097	0.0052	0.0107	0.0050	**0.0050**	0.0026	0.0123	0.0070

Table 5.3: Results of rotational relative pose error (RPE) (m). RDS-SLAM (1) and (3) are evaluated results only using keyframes.

Seq.	ORB SLAM3		DS-SLAM		DynaSLAM		SLAM-PCD		RDS-SLAM (1) KeyFrame (Mask R-CNN)		RDS-SLAM (2) All (Mask R-CNN)		RDS-SLAM (3) KeyFrame (SegNet)		RDS-SLAM (4) All (SegNet)	
	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.
w/half	7.2352	5.9487	0.8142	0.4101	0.7842	0.4012	0.7440	0.3459	0.8194	0.4858	0.8216	0.4347	**0.7332**	0.3712	1.8828	1.5250
w/rpy	8.7683	6.4583	3.0042	2.3065	0.9894	0.5701	1.3831	0.8318	1.4736	1.0262	9.3192	8.5720	**0.4973**	0.2586	13.1693	12.0103
w/static	6.0054	5.5995	0.2690	0.1215	**0.2612**	0.1259	0.2631	0.1119	1.4966	1.2839	1.1686	0.9917	0.4944	0.3112	0.4944	0.3112
w/xyz	7.8974	5.5917	0.8266	0.2826	0.6284	0.3848	**0.6227**	0.3807	0.7739	0.4943	0.7236	0.4435	0.7768	0.4886	0.9222	0.6509
s/static	0.3007	0.1300	0.2735	0.1215	0.3416	0.1642	0.2782	0.1210	0.3217	0.1522	0.3091	0.1325	**0.1520**	0.0821	0.3338	0.1706

5.6 Experimental Results

We evaluate the tracking accuracy using TUM [49] indoor dataset and demonstrate the real-time performance by comparing with state-of-the-art vSLAMs methods using, when possible, the results in the original papers.

5.6.1 System Setup

RDS-SLAM is evaluated using GeForce RTX 2080Ti GPU, Cuda 11.1, and docker [3]. Docker is used to deploy different kinds of semantic segmentation methods on the same machine. We also use Kinect v2 [4] camera to evaluate in real environment.

5.6.2 Tracking Accuracy Evaluation

The proposed method was compared against the ORB-SLAM3 and similar semantic-based algorithms to quantify the tracking performance of our proposal in dynamic scenarios.

The TUM RGB-D dataset contains color and depth images along the ground-truth trajectory of the sensor. In the sequence named "*fr3/walking_ **" (labeled as f3/w/*), two people walk through an office. This is intended to evaluate the robustness of vSLAM in the case of quickly moving dynamic objects in large parts of a visible scene. Four types of camera motion are included in *walking* data sequences: 1) "*xyz*", the Asus Xtion camera is manually moved along three directions (xyz); 2) "*static*", where the camera is kept in place manually; 3) "*halfsphere*", where the camera is moved on a small half-sphere of approximately one-meter diameter; 4) "*rpy*", where the camera is rotated along the principal axes *(roll-pitch-yaw)*. In the experiment, the person is dealt with as the only priori dynamic object.

We compared the trajectory of camera with ORB-SLAM3 [5], DS-SLAM [6], and

[3] `https://docs.docker.com/`
[4] `https://github.com/code-iai/iai_kinect2`
[5] `https://github.com/UZ-SLAMLab/ORB_SLAM3`
[6] `https://github.com/ivipsourcecode/DS-SLAM`

DynaSLAM [7]. Fig. 5-14 compares the obtained trajectories using their source codes and therefore the trajectories are not exactly the same as the ones in their original paper. We evaluated RDS-SLAM using both Mask R-CNN (M) and SegNet (S). The trajectory of DynaSLAM that use Mask R-CNN is very similar with our Mask R-CNN version as shown in Fig. 5-14 (m-p) and Fig. 5-14 (q-t). The performance of our SegNet version (Fig. 5-14 (i and j)) is similar to the DS-SLAM (Fig. 5-14 (e and f)).

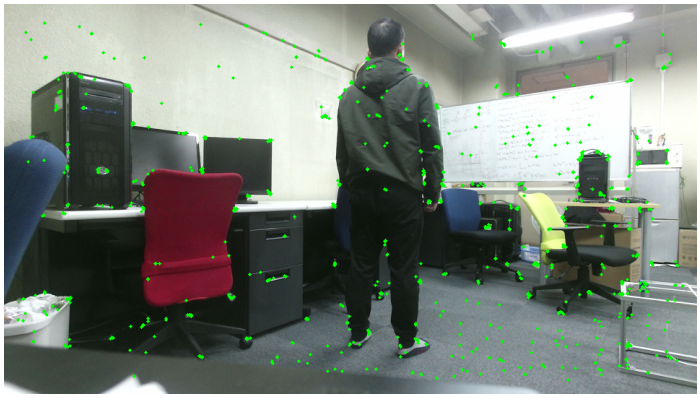
The error in the estimated trajectory was calculated by comparing it with the ground truth, using two prominent measurements: absolute trajectory error (ATE) and relative pose error (RPE) [49], which are well-suited for measuring the performance of the vSLAM. The root mean squared error (RMSE), and the standard deviation (S.D.) of ATE and RPE are compared. Each sequence was run at least five times as dynamic objects are prone to increase the non-deterministic effect. We compared our method with ORB-SLAM3 [18], DS-SLAM [43], DynaSLAM [41], SLAM-PCD [47], DM-SLAM [46], and Detect-SLAM [44]. The comparison results are summarized in Tables 5.1, 5.2, and 5.3. DynaSLAM reported they obtained the best performance using the combination of Mask R-CNN and geometric model. We mainly focus on the time cost problem caused by semantic segmentation. Contrary to the very heavy geometric model that DynaSLAM used, we only use the very light geometric check, such as RANSAC, photometric error to deal with the outliers that not rely on the priori dynamic objects.

Our proposal outperforms the original ORB-SLAM3 (RGB-D mode only without IMU) and obtains similar performance with DynaSLAM, SLAM-PCD, and DM-SLAM, in which the tracking error is already very small. Different from them, we use the non-blocked model. The first few frames do not have any semantic information. The number of keyframes that have a semantic label is smaller than suing the blocked model because the processing speed of the tracking thread is much faster than the semantic segmentation (especially for the heavy model, Mask R-CNN). However, we achieved a similar tracking performance using less semantic information.

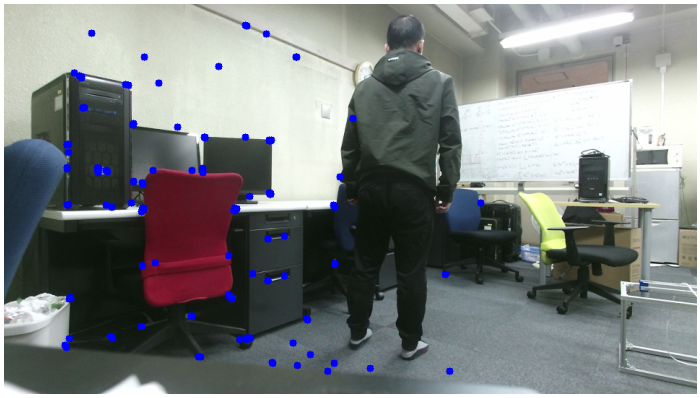
[7] `https://github.com/BertaBescos/DynaSLAM`

Table 5.4: The execution time comparison of TUM Dataset. We use the data in their original paper as possible. If not provide in their papers, we approximate the processing time.

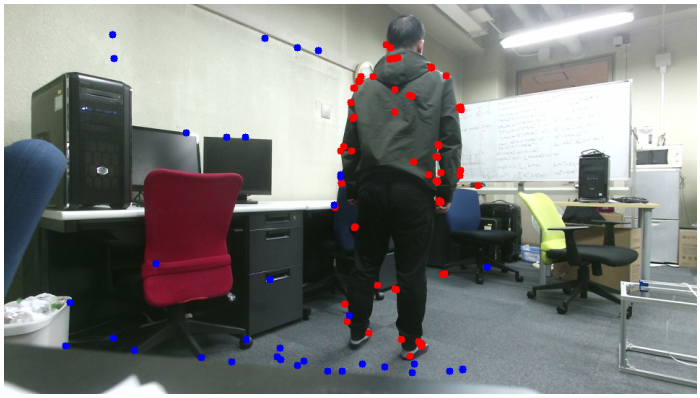
Method	GPU	Semantic	Segmentation/Detection Time (ms)	Time of Other Models Related to Tracking (ms)		Track Each Frame (ms)
ORB-SLAM3	-	-	-	-		22 - 30
Detect-SLAM	Nvidia GPU GTX960M	SSD	310	Propagation: Updating:	20 10	> 310
DS-SLAM	P4000	SegNet	37.57330	ORB feature extraction: Moving consistency check:	9.375046 29.50869	> 65
DynaSLAM	Nvidia Tesla M40 GPU	Mask R-CNN	195	Multi-view Geometry: Background Inpainting:	235.98 (w/rpy) 183.56 (w/rpy)	> 300
DM-SLAM	GeForce GTX 1080 Ti	Mask R-CNN	201.02	Ego-motion: Dynamic Point Detection:	3.16 40.64	> 201
RDS-SLAM (1) TUM	GeForce RTX 2080Ti	Mask R-CNN	200	Mask Generation: Update Moving Probability: Semantic-based Optimization:	5.42 0.17 0.54	50 - 65 (15HZ)
RDS-SLAM (2) TUM	GeForce RTX 2080Ti	SegNet	30	Mask Generation: Update Moving Probability: Semantic-based Optimization:	5.04 0.17 0.50	50 - 65 (15HZ)
RDS-SLAM (3) Kinect v2	GeForce RTX 2080Ti	Mask R-CNN	200	Mask Generation: Update Moving Probability: Semantic-based Optimization:	9.01 0.15 0.38	22 - 30 (30HZ)
RDS-SLAM (4) Kinect v2	GeForce RTX 2080Ti	SegNet	30	Mask Generation: Update Moving Probability: Semantic-based Optimization:	9.05 0.18 0.45	22 - 30 (30HZ)



(a) Initial features



(b) After tracking last frame



(c) After tracking local map

Figure 5-15: Result of real environment. The green features are in initial status and their moving probability is 0.5. The blue features are static features and the red are outliers. (a) is the original detected ORB features. (b) is the output after tracking last frame process and (c) is the result after tracking local map process.

Table 5.5: Semantic Keyframe Number Comparison (Mask R-CNN).

	Total Frames	15HZ		30HZ	
		Total Time (s)	Semantic KeyFrame Num.	Total Time (s)	Semantic KeyFrame Num.
w/xyz	859	57.3	286	28.6	143
w/rpy	910	60.7	303	30.3	151
w/half	1067	71.1	355	35.6	178
w/sit	707	47.1	235	23.6	118

5.6.3 Real Environment Evaluation

We tested RDS-SLAM using Kinect2 RGB-D camera, as shown in Fig. 5-15. All the features are in initial status when in the first few frames because they have not yet obtained any semantic information. The static features will be increasingly detected over time and used to estimate camera pose. The features on the person is detected and excluded from tracking. The algorithm runs in around 30HZ, as shown in Table 5.4.

5.6.4 Execution Time

Tab. 5.4 compares the execution time of vSLAM algorithms. In the blocked model, the tracking thread needs to wait for the semantic label. The speed of the other methods is related to the semantic segmentation methods used. The heavy the semantic model used, the higher the total time consuming is. Although DynaSLAM achieved good tracking performance, the processing time is long due to Mask R-CNN. As we known, DynaSLAM is not a real-time algorithm. DS-SLAM is the second fastest algorithm because it uses a lightweight semantic segmentation method, SegNet. However, the architecture used is also a blocked model. The execution time will increase if a more time-consuming method is used. Our method uses the non-blocked model and runs almost at a constant speed regardless of the segmentation methods.

We evaluate the error metric of TUM dataset using 15HZ by manually adding some time delay in the tracking thread because TUM dataset is very short. Very small semantic information can be obtained in this short time. We compare the time and the number of keyframes that obtained semantic label (Semantic keyframe

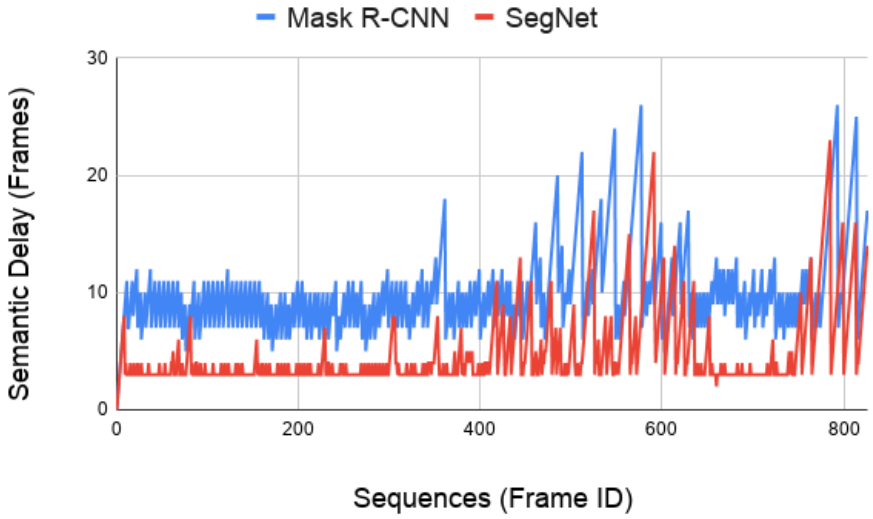


Figure 5-16: Semantic Delay of TUM w/xyz Dataset. The average value of Mask R-CNN case is 10 and SegNet is 5.

Number) in Tab. 5.5. We only compared the Mask R-CNN version because SegNet is faster and it can segment almost all the keyframes in each dataset. We assume the time cost of Mask R-CNN is 0.2s for segmenting each frame. The total time of running the fr3/w/xyz dataset is about 57.3s for 15HZ, however, only 28.3s for 30HZ. In this short time, the number of semantic keyframes in 30HZ (143) is two times smaller than 15HZ (286). Usually, the more keyframes are segmented, the better tracking accuracy can be achieved. This depends on the specific application and the segmentation methods used.

In the bi-direction model, we selected two keyframes at the same time. We offered two strategies to segment them: 1) infer images at the same time as a batch on the same GPU, 2) infer images on the same GPU sequentially (one by one). We suggest using (1) if the GPU can infer a batch of images at the same time. Our Mask R-CNN version uses (1) because we found we need 0.3s-0.4s in case (1) and 0.2s in case (2). Our SegNet version is evaluated using the strategy (2) because SegNet is very fast and can be segmented sequentially.

5.6.5 Semantic Delay Evaluation

We have analyzed the semantic delay by assuming the keyframe is selected every two frames (see Fig. 5-6). In experiment, we follow the keyframe selection policy used in ORB SLAM3 and we compared the semantic delay of Mask R-CNN case and SegNet case using the TUM dataset, as shown in Fig. 5-16. The semantic delay is influenced by these factors: 1) the segmentation speed, 2) the keyframe selection policy, 3) the undetermined influence caused by the different running speed of multiple threads (e.g., Loop Closing thread), 3) the hardware configures. In the fr3/w/xyz dataset, the camera sometimes moves very slow and sometimes moves forward or backward. As a result, this will change the keyframe selection frequency and cause the variance of semantic delay.

5.7 Conclusions

A novel vSLAM system, semantic-based real-time visual SLAM (RDS-SLAM) for the dynamic environment using an RGB-D camera is presented. We modify ORB-SLAM3 and add a semantic tracking thread and a semantic-based optimization thread to remove the influence of dynamic objects using semantic information. These new threads run in parallel with the tracking thread and therefore, the tracking thread is not blocked to wait for semantic information. We proposed a keyframe selection strategy for semantic segmentation to obtain as the latest semantic information as possible that can deal with segmentation methods with different speeds. We update and propagate semantic information using the moving probability which is used to detect and remove outliers from tracking using a data association algorithm. We evaluated the tracking performance and the processing time using the TUM dataset. The comparison against state-of-the-art vSLAMs shows that our method achieved good tracking performance and can track each frame in real-time. The fastest speed of the system is about 30HZ, which is similar to the tracking speed of ORB-SLAM3.

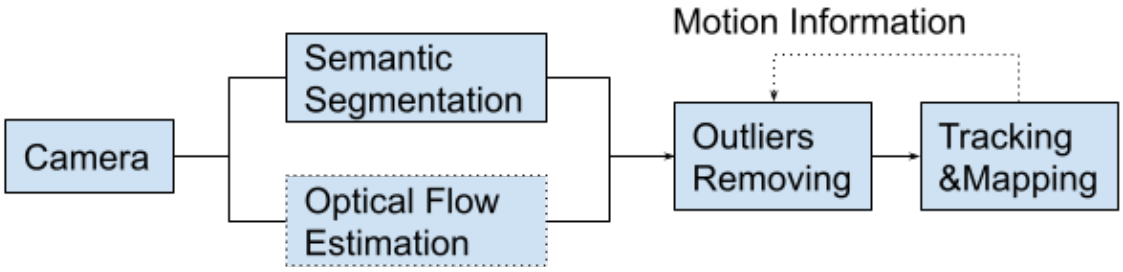
Chapter 6

RDMO-SLAM

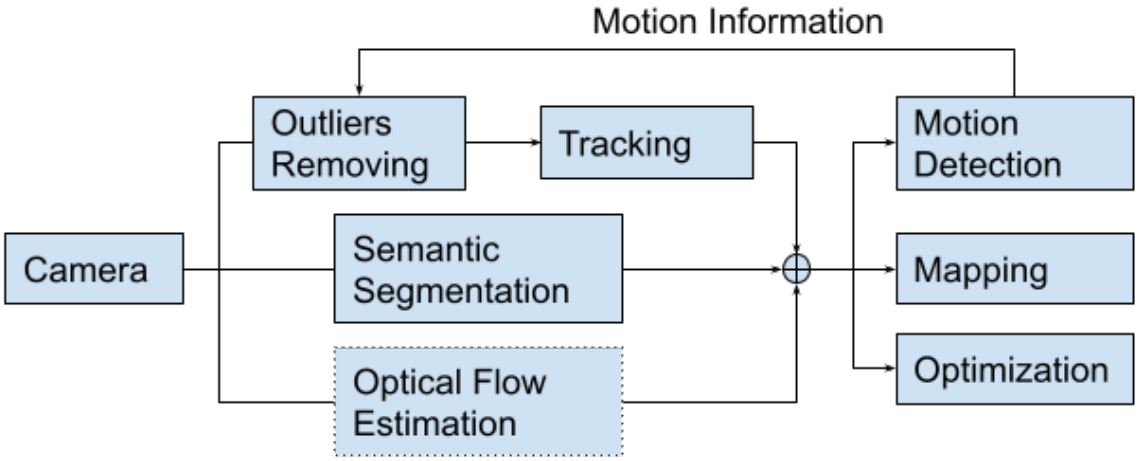
In the previous chapter, we introduced a real-time vSLAM based on the non-blocked model, RDS-SLAM, which isolates tracking and semantic segmentation by adding a semantic thread and moving probability estimation. However, Mask R-CNN, one of the popular semantic segmentation methods, only supplies a small amount of semantic information because only a few keyframes can be segmented within a short time. In this chapter, we propose a novel vSLAM, RDMO-SLAM [14], which can leverage more semantic information while ensuring the real-time nature by adding semantic label prediction using dense optical flow. Besides, we also estimate the velocity of each landmark and use them as constraints to reduce the influence of dynamic objects in tracking. Demonstrations are presented, which compare the proposed method to comparable state-of-the-art approaches using dynamic sequences. We improved the real-time performance from 15 Hz (RDS-SLAM) to 30 Hz while keeping robust tracking in dynamic scenes.

Many studies try to eliminate or reduce the influence of dynamic objects using various segmentation methods. For example, Detect-SLAM [44] uses the object detection (SSD [45]) approach to improve tracking performance; similarly, DynaSLAM [41] and DM-SLAM [46] use Mask R-CNN [24], DS-SLAM [43] uses SegNet [22], and KMOP [12] uses Open-Pose [28] and k-means [25].

The execution speeds of these methods are limited by the semantic model used, e.g., Mask R-CNN, SSD, and OpenPose, because these methods need to wait for the



(a) Blocked model.



(b) Non-blocked model.

Figure 6-1: Blocked model vs non-blocked model. The optical flow model is optional. The semantic model can be Mask R-CNN, SegNet, SSD or others. Segmentation and optical flow can run in parallel for frames or keyframes. The feedback, motion information, is optional for the blocked model.

semantic result/information, e.g., label, bounding box, before tracking. Such an architecture is called a *blocked model*, as shown in Fig. 6-1 (a). In the previous chapter, we proposed a novel real-time vSLAM architecture, RDS-SLAM [13], validated using both Mask R-CNN and SegNet using *non-blocked model*, as shown in Fig. 6-1 (b). RDS-SLAM can guarantee the speed of tracking free from speed limitation of semantic models using multi-thread and moving probability estimation of each map point. However, it has some shortcomings: a) Mask R-CNN is slow and cannot segment every keyframe to use more semantic information in a limited time. It cannot run stably at 30 Hz using the TUM [49] dataset because insufficient semantic information obtained at 30 Hz for some of the dynamic scenes of the TUM dataset, which are very

short, only about half a minute; b) only predefined objects trained by Mask R-CNN are handled. We try to ensure real-time performance (30 Hz) while keeping robust tracking by exploiting optical flow.

One critical challenge of using semantic information is time complexity. Although there are some lightweight semantic segmentation models, e.g., SegNet, the total time of tracking one frame is still more than the original ORB-SLAM3. Besides, sometimes a complex CNN architecture is required for robots to perform high-level tasks, e.g., human-robot interaction and semantic mapping. To acquire more semantic information in a limited time, we use dense optical flow for each pixel to predict the semantic label of Mask R-CNN. Another challenge is that only predefined objects trained by CNN are used to judged outliers. Optical flow can estimate the pattern of motion for every feature, including features on the undefined objects. The velocity of map points can be estimated by optical flow and used as a constraint to reduce the influence of outliers from the tracking process.

We propose a semantic label prediction algorithm to generate more semantic information for the keyframes that are not segmented. Real-time tracking under dynamic environments is achieved while keeping robust tracking using a heavy CNN architecture. Besides, the velocity of landmarks is estimated with the aid of scene flow and Kalman filter. These two constraints (the semantic label and velocity) can reduce the influence of dynamic objects in vSLAM.

The main contributions of RDMO-SLAM are as follows.

(1) We propose a novel semantic-based real-time vSLAM algorithm using Mask R-CNN and PWC-Net for dynamic environments, RDMO-SLAM, an extension of RDS-SLAM, which can achieve both good tracking performance and the real-time nature.

(2) We predict the semantic result of Mask R-CNN using optical flow to obtain more semantic information so that the tracking thread uses as much semantic information as possible.

(3) We demonstrate the real-time performance (30 Hz) under dynamic environments using the TUM dataset and an AR demo in the case of using a heavy CNN

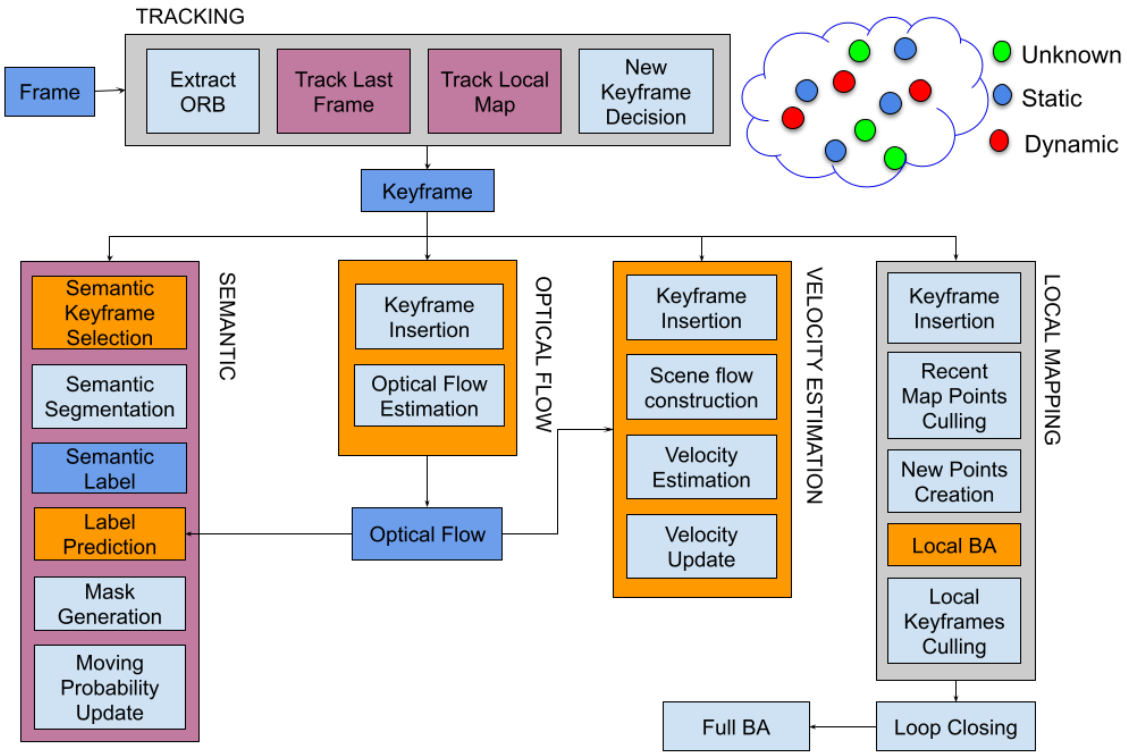


Figure 6-2: System architecture. Models with orange color are the ones that are modified from RDS-SLAM or new blocks. Models with magenta color are derived from RDS-SLAM but different from ORB-SLAM3. Blocks in blue are important data structures.

architecture, Mask R-CNN.

6.1 System Overview

Fig. 6-2 shows the architecture of RDMO-SLAM, which is implemented based on ORB-SLAM3 and RDS-SLAM. There are four threads in ORB-SLAM3: tracking, local mapping, loop closing, and full BA. In RDS-SLAM, we add a semantic thread to request the semantic information and update the moving probability of map points into ATLAS. We classify these landmarks into three subsets, unknown, static, and dynamic according to their moving probability, and then use as many static ones as possible in the tracking thread. We follow the basic idea of RDS-SLAM, add two new threads, optical flow, and velocity estimation threads, and modify some modules of

RDS-SLAM and ORB-SLAM3.

The tracking thread aims to estimate the initial camera pose via feature matching and select keyframes used by the local mapping thread to update the map and further optimize the pose estimation via BA. In the semantic thread, first, we request semantic labels of selected keyframes, then generate mask images of predefined dynamic objects, and finally calculate as well as update the moving probability of map points in the global map using semantic information. Different from RDS-SLAM, we add a new module called *Label Prediction*, which is designed to predict semantic labels using optical flow while waiting for the semantic result. In the optical flow thread, we estimate the dense optical flow for each keyframe and use the optical flow to predict the semantic label and estimate the scene flow of landmarks. The velocity estimation thread aims to calculate and update the velocity of map points using the scene flow of map points. The velocity of landmarks is used as another constraint to filter bad data associations from tracking. Finally, this semantic information expressed by the moving probability and the velocity of landmarks is used to filter the outliers.

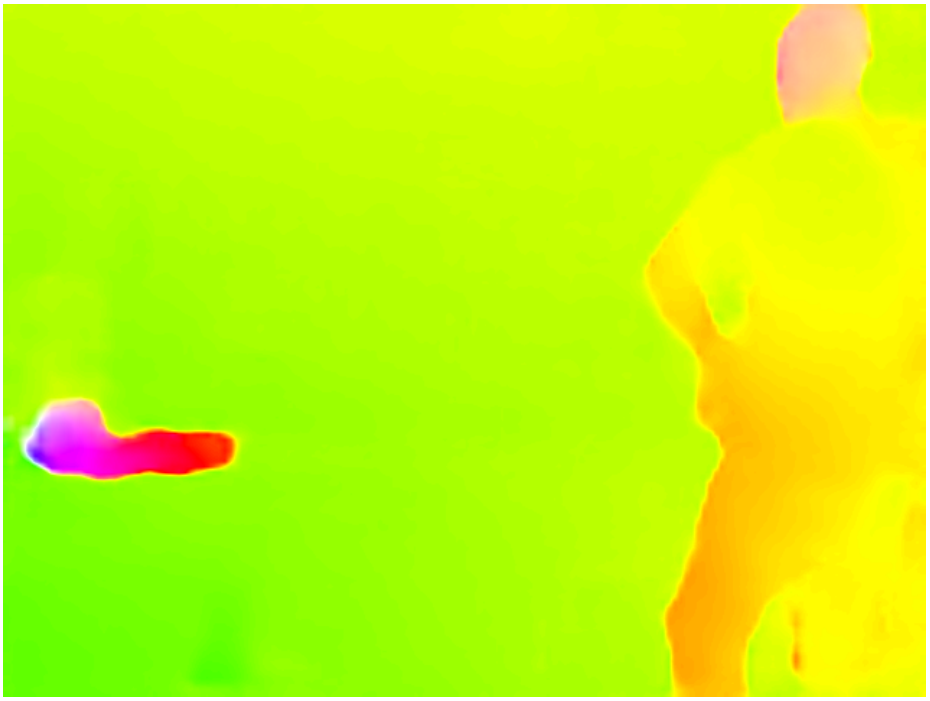
6.2 Optical Flow Thread

Optical flow estimation is a basic computer vision problem and has many applications, such as autonomous driving, multi-object tracking, and vSLAM. PWC-Net [63] is a compact but effective CNN model for optical flow estimations. It adopts a fast, scalable, and end-to-end trainable CNN framework [64]. It is designed using well-established principles, pyramidal processing, warping, and the use of a cost volume. It warps the CNN features of the second image toward the first image. Then uses the warped features and the features of the first image to construct a cost volume, which is processed by a CNN to estimate the optical flow. PWC-Net is more lightweight and easier to train than the recent FlowNet2 [65] model and can run at about 35 fps [63] on Sintel [66] resolution (1024 x 436).

Each pixel of optical flow result stores two float values $F = (f_x, f_y) \in \mathbb{R}^2$, which indicate the displacement of each pixel between a previous and current image. For-

(a) Previous RGB image (10)

(b) Current RGB image (11)



(c) Optical flow pattern

Figure 6-3: Optical flow estimation example using the TUM dataset. (a) and (b) are images from consecutive keyframes, and (c) is their optical flow visualized using HSV color constructed by flow direction and flow magnitude.

mally, for each pixel (x_1, y_1) in the previous image, the corresponding pixel in the current image is given by:

$$(x_2, y_2)^T = (x_1 + f_x, y_1 + f_y)^T. \tag{6.1}$$

A ROS version of PWC-Net[1] (Caffe models[2]) is used to predict the optical flow for

[1] `https://github.com/ActiveIntelligentSystemsLab/pwc_net_ros`

[2] `https://github.com/NVlabs/PWC-Net/blob/master/Caffe/model/pwc_net.caffemodel`

each pixel of consecutive keyframes. The input is the consecutive two RGB images, and the output is the optical flow, as shown in Fig. 6-3. Optical flow can only detect the motion part of the body, e.g., hand and leg. However, the unstable features on the static parts of the body cannot be detected. This problem can be solved together using semantic segmentation.

Later, we use the result of optical flow to predict the semantic label of keyframes in the semantic thread to increase the speed of semantic information generation. Besides, the result is also used by velocity thread to calculate the velocity of map points.

6.3 Semantic Thread

This thread aims to provide semantic information and use them to update the moving probability of map points. Fig. 6-2 (semantic modules) shows the general flow. First, we select one keyframe to request a semantic label using Mask R-CNN. However, it requires a very long time (about 200ms) to obtain the semantic result/label. To obtain more semantic information, we propose an algorithm to predict the semantic label of the keyframes using the previous obtained semantic label and optical flow patterns of the reference keyframes, while waiting for the result of the current semantic request. After obtaining the semantic label, we generate a mask of dynamic objects, which will be used to update the moving probability. We will explain in detail in the following sub-sections.

6.3.1 Semantic Keyframe Selection

The *semantic delay* [13] between the semantic and tracking threads will increase over time if all keyframes are segmented sequentially. The tracking thread cannot obtain the latest and enough semantic information in real-time. To decrease the semantic delay, only the keyframes from the front and back of the keyframe queue KF are selected to request semantic labels in RDS-SLAM. However, this will cause many keyframes not able to obtain semantic results. In other words, not all the keyframes

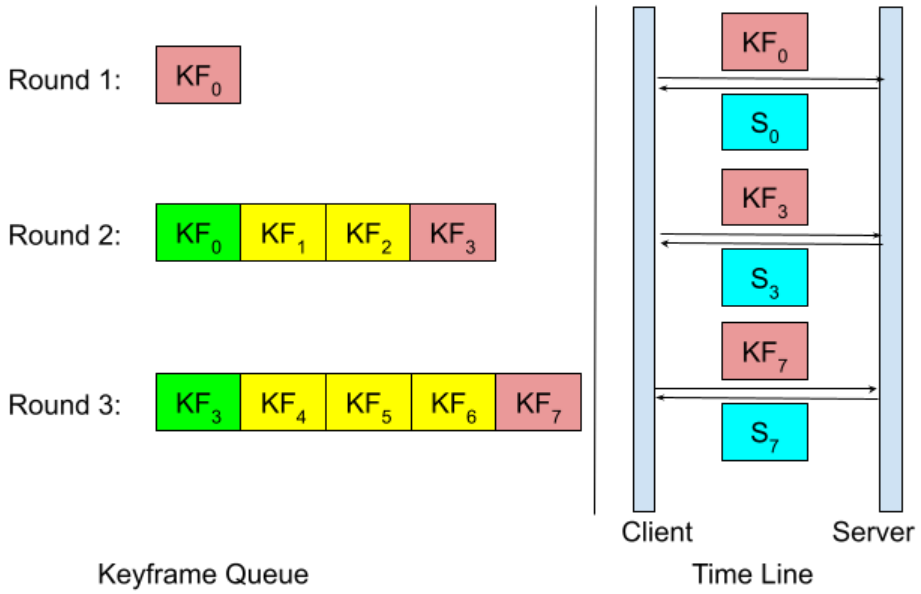


Figure 6-4: Semantic timeline. The left side is the contents inside the keyframe queue KF, and the right side is the timeline of requesting semantic labels. $S_{(.)}$ is the semantic label returned from the semantic server. The keyframes in yellow are the ones that need to predict. The keyframes in pink are the ones that request the semantic label from the semantic server, and those in green are the ones that have obtained semantic results in the previous rounds.

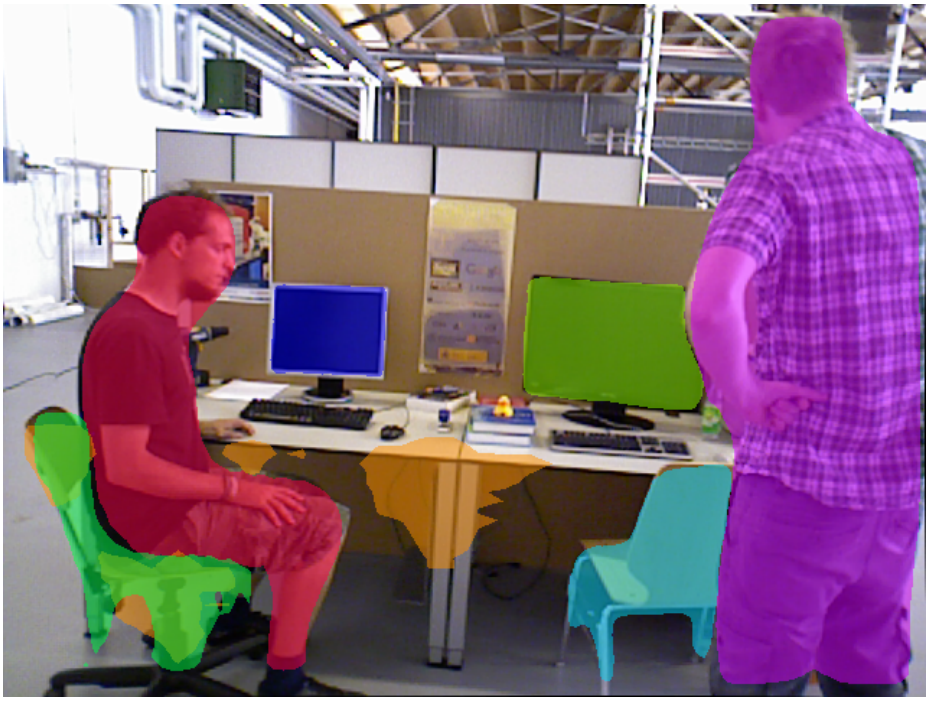
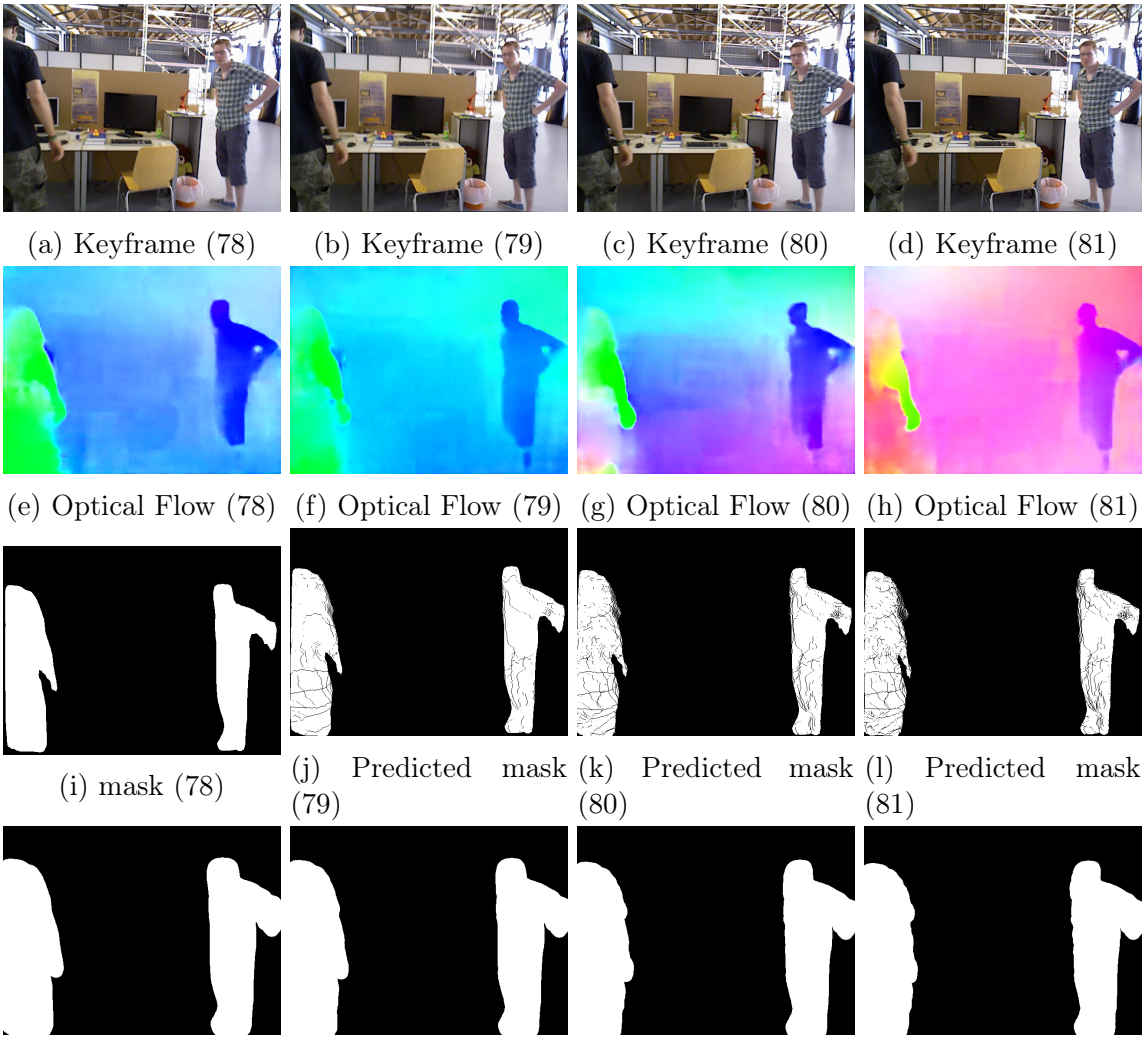


Figure 6-5: Semantic segmentation result.



(a) Keyframe (78) (b) Keyframe (79) (c) Keyframe (80) (d) Keyframe (81)

(e) Optical Flow (78) (f) Optical Flow (79) (g) Optical Flow (80) (h) Optical Flow (81)

(i) mask (78) (j) Predicted mask (79) (k) Predicted mask (80) (l) Predicted mask (81)

(m) Dilated mask (78) (n) Dilated mask (79) (o) Dilated mask (80) (p) Dilated mask (81)

Figure 6-6: Predicted mask. (a) is the reference keyframe, and (b)-(d) are the keyframes that need to predict.

can have the chance to get a semantic result. This may result in non-robust or unstable tracking under complex environments. RDS-SLAM has been only evaluated at 15 Hz rather than 30 Hz in TUM dataset because adequate semantic information cannot be obtained within a short time using Mask R-CNN. To handle this drawback, we try to ensure that almost all keyframes can obtain semantic labels. We always select the latest keyframe from the back of the KF queue to request semantic results.

Fig. 6-4 shows an example of keyframe selection policy. In round 1, we select the first keyframe KF_0 to request a semantic label from a semantic server (Mask

R-CNN). In round 2, we select the latest keyframe (KF_3) in the queue to request semantic labels. We predict the semantic label for others ($KF_1 - KF_2$). Similarly, in the next round, we take the element KF_7 from the back of the queue to request and predict the others sequentially (KF_4 - KF_6).

6.3.2 Semantic Segmentation

We use Mask R-CNN[3] trained with the MS COCO [61] dataset as the semantic server. Fig. 6-5 shows an example of a semantic segmentation result. However, the semantic segmentation result is not always correct, and the edge of the object is difficult to classify. Besides, only pretrained objects can be segmented. Therefore, we dilate the mask to cover the features on the boundary and use the velocity of map points as another constraint to remedy this insufficiency.

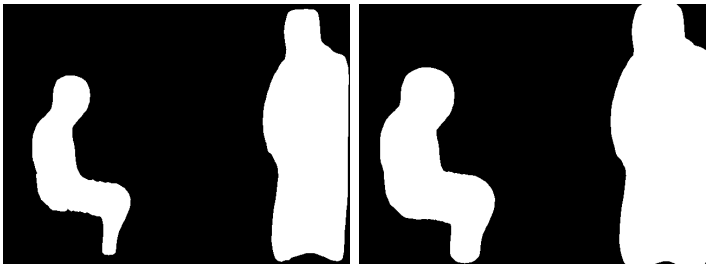
6.3.3 Semantic Label Prediction

To ensure that more keyframes can obtain the semantic label, we predict the semantic labels for not-yet-segmented keyframes using optical flow. As shown in Fig. 6-4 (round 2), KF_0 is the reference keyframe that has already been segmented, KF_3 the current request, and $KF_1 - KF_2$ are the predicted ones using the reference keyframe. Given a reference keyframe label $I_r(x_r, y_r)$ and the corresponding optical flow vector (f_x, f_y), the predicted label $I_p(x_p, y_p)$ is calculated as follows:

$$I_p(x_p, y_p) = I_p(x_r + f_x, y_r + f_y) = I_r(x_r, y_r). \tag{6.2}$$

Fig. 6-6 shows an example of semantic label prediction. From the label of the reference keyframe (a) and the optical flow (f-h), we predict semantic labels of the subsequent keyframes and generate their mask images, as shown in (j-l).

[3] `https://github.com/matterport/Mask_RCNN`



(a) original mask (b) dilated mask

(c) before dilation

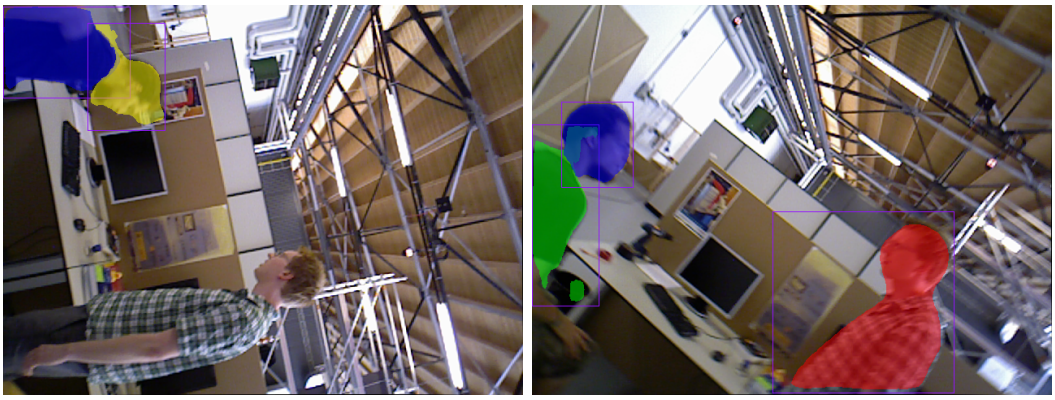
(d) after dilation

Figure 6-7: Dilation example. Features in red are outliers after dilation operation, and in blue are the observed static features.

6.3.4 Semantic Mask Generation

We generate mask images of predefined dynamic objects such as persons and animals by applying dilation operation to the predicted semantic labels to fill the holes and

Figure 6-8: Moving probability. θ_s and θ_d are threshold values.



(a) (735) w/rpy (b) (766) w/rpy

Figure 6-9: Segmentation accuracy is not correct in the case of a large camera rotation. (a) the right person is not segmented, and the head of the left person is wrongly segmented. (b) the head of the left person is wrongly segmented.

expand object boundaries. As shown in Fig. 6-7, since the features around the boundary of dynamic objects can also be the outliers, they will be covered after dilating the mask. The noise or holes on the predicted labels can also be smoothened, as shown in Figs. 6-6 (n-p).

6.3.5 Moving Probability Update

We define the moving probability $p(m_t^j)$, $m_t^j \in M$ where $M = \{static\,(s), dynamic\,(d))\}$, for a map point j that matches with features in the keyframe, as shown in Fig. 6-8 [13] . We omit the superscript j in the following derivation. We update the moving probability in the semantic thread using Bayesian filter [62] as follows:

$$\begin{aligned} bel(m_t) &= p(m_t|z_{1:t}, m_0) \\ &= \eta p(z_t|m_t) \int p(m_t|m_{t-1}) bel(m_{t-1}) dm_{t-1}, \end{aligned} \tag{6.3}$$

where $\eta = 1/(bel(m_t = d)+bel(m_t = s))$ and $p(m_0) = 0.5$ is the initial probability and $p(z_t|m_t)$ is the observation likelihood, which is set according to the semantic label. It is reasonable to assume that the current observation is independent of the previous ones. Thus, we define the observation model as follows:

$$p(z_t = d|m_t = d) = \alpha, \tag{6.4}$$

$$p(z_t = s|m_t = s) = \beta, \tag{6.5}$$

where α is given a fixed value in RDS-SLAM. Usually, the segmentation accuracy is influenced by the camera rotation around the optical axis, as shown in Fig. 6-9, if CNN is not trained using enough data for such cases.

The rotation of the camera is presented as follows:

$$R(r) = rot(T(\xi)) = exp(r\hat{}), \tag{6.6}$$

where, r is the Euler angle (roll, pitch, yaw) and $r\hat{} \in so(3)$. We heuristically adjust the reliability of semantic segmentation (α) according to the roll component and set α to a small value when the rotation is huge. We set α according to the roll component when it is greater than a threshold γ.

$$\alpha = \begin{cases} max\{min\{0.9, \frac{1}{exp^{(||roll(r)||-2)}}\}, 0.1\} & ||roll(r)|| > \gamma \\ 0.9 & others. \end{cases} \tag{6.7}$$

In our experiment, γ is set to 1.5 to omit the relatively small camera rotation and β to 0.9 by assuming the observation is fairly robust for static objects.

6.3.6 Algorithm Implementation

Alg. 10 shows the detailed implementation of semantic thread. To maintain the information exchange of optical flow and semantic segmentation threads, checking

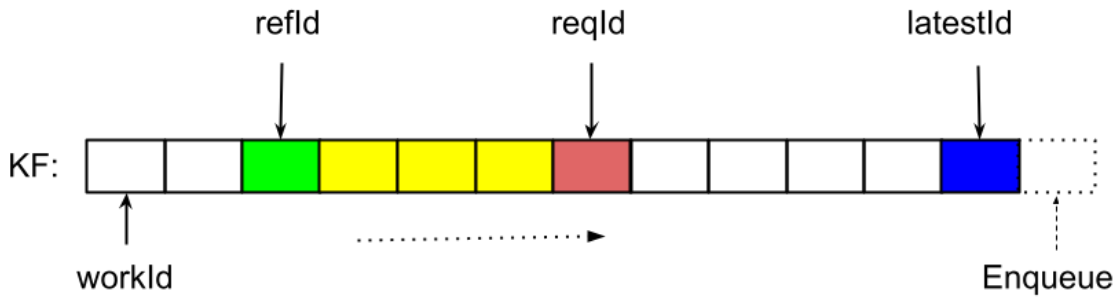


Figure 6-10: Semantic Prediction Algorithm. The $workId$ is the current working pointer that walks through every keyframe sequentially. The $refId$ is the id of the reference keyframe, and $reqId$ is the id of the last semantic request. $lstestId$ is the id of the latest keyframe by now.

functions "IsOpticalFlowReady()" and "IsSemanticReady()" are used respectively to sync the data flow. To handle each keyframe incrementally, we designed some indicators/pointers to control the flow of the algorithm, as shown in Fig. 6-10. The keyframes in yellow that need to predict are located between the reference keyframe ($refId$) and the last semantic request keyframe ($reqId$). First, we take out the current keyframe (line 5) and the latest keyframe (lines 6-7) from the back of the KF queue. Then, we segment the first few keyframes ($initNum$), as shown in Alg. 10 (lines 8-17) considering some datasets are short. The tracking is blocked to wait for the semantic results for these keyframes. Besides, it will consume more time (>300ms in our experiment) to segment the first image due to the GPU initialization. Therefore, we suggest waiting for the segmentation result of the first few keyframes. In the experiment, $initNum$ is set to 1 when evaluating the TUM dataset and 0 for a real camera. Lines 21-25 are to select the latest keyframe to request the semantic label non-blocked when 1) the last request $reqKF$ has already obtained the semantic label and 2) there are new elements in the KF queue waiting to segment. Lines 26-39 are to predict the semantic labels using the selected reference keyframe that have already obtained the semantic result (lines 26-28), and the keyframes that need to predict have already got the optical flow (lines 31-33). The keyframe is predicted when the semantic segmentation processing speed is slower than the new keyframe enqueuing speed. This algorithm predicts semantic labels while waiting for the semantic label.

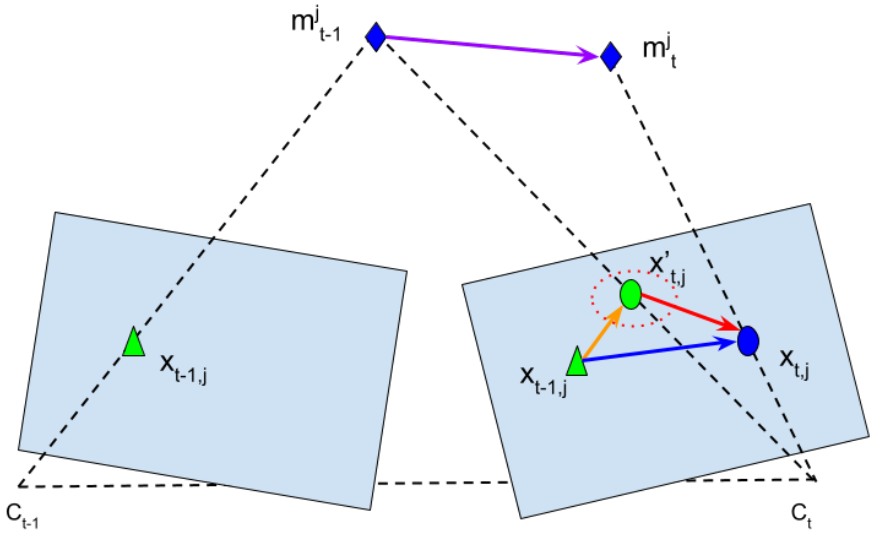


Figure 6-11: Optical and scene flows. The blue vector is the original optical flow vector, orange is the rigid flow, and red is the non-rigid flow. The purple vector is a scene flow vector.

We wait for the segmentation result (lines 18-20) when 1) no new enqueued keyframe exist, or 2) all the keyframes before the last request $reqId$ are already handled.

We update the semantic information after the semantic label is obtained either by semantic segmentation or prediction, as shown in Alg. 11. Similarly to RDS-SLAM, we generate the mask images of dynamic objects and update the moving probability of map points using the generated mask.

6.4 Velocity Estimation Thread

Semantic segmentation can only handle predefined dynamic objects, and the segmentation is not always accurate. We add velocity constraints for objects to further reduce the influence of outliers.

As shown in Fig. 6-11, given a pixel $x_{t,j}$ in the previous image, we can estimate the corresponding pixel in the next image using

$$x_{t,j} = x_{t-1,j} + F_{x_{t,j}}, \tag{6.8}$$

where $F_{x_{t,j}}$ is a optical flow vector shown in Fig. 6-11 (blue vector) and $x'_{t,j}$ is the estimated point using the camera motion assuming the camera is the only moving object. The motion of the camera needs to be subtracted from the optical flow. Then, the sparse scene flow of landmarks is calculated by

$$\begin{aligned} s &= m_t - m_{t-1} && (6.9)\\ &= \pi^{-1}(x_{t,j}, D(x_{t,j}), T_t^w(\xi)) && (6.10)\\ &- \pi^{-1}(x_{t-1,j}, D(x_{t-1,j}), T_{t-1}^w(\xi)), \end{aligned}$$

where π^{-1} is a function that back project one 2D point in the image to the 3D world space using the camera pose and depth image.

The velocity of each map point is calculated by:

$$z_t = \frac{\|s\|}{\Delta t}, \tag{6.11}$$

where Δt is the time difference between consecutive keyframes.

Some velocities are very large due to inaccurate camera pose estimation, inaccurate depth data, and wrongly matched feature points. Besides, the features are extracted from different pyramid layers of the image. This also results in inaccurate or wrong velocity estimation. We update the velocity using Kalman Filter [62] by:

$$\begin{aligned} \bar{v}_t &= v_{t-1}, && (6.12)\\ v_t &= \bar{v}_t + K_t(z_t - \bar{v}_t), && (6.13) \end{aligned}$$

where K_t is the Kalman gain and z_t is the newly calculated velocity. We assume the map points move at a constant speed. The predicted velocity $\bar{v}_t$ is equal to the previous speed. Ideally, the speed of static map points should be nearly zero.

We use the velocity of map points as another constraint to further filter outliers. As we knew, it is difficult to find an optimal threshold to judge outliers. Not enough features may be left if the threshold is set too small. In our view, there is no close

form to decide the optimal value for all the frames or scenes. In our experiment, we set a large value to remove only obvious outliers with very large velocity.

6.5 Tracking

To let vSLAM run in real-time, we separated the semantic thread and the velocity estimation thread from the tracking thread, so as not to block the tracking. The moving probability and the velocity of landmarks are stored in the map. We use them as constraints to filter outliers from camera ego-motion estimation.

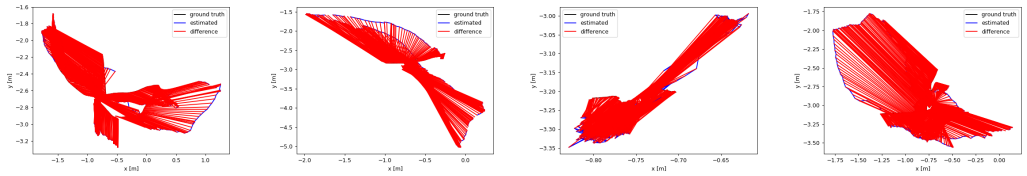
As shown in Fig. 6-8, we judge the status of objects using

$$Status(m_t) = \begin{cases} dynamic & bel(m_t) > \theta_d \\ static & bel(m_t) < \theta_s \\ unknown & others \end{cases} . \tag{6.14}$$

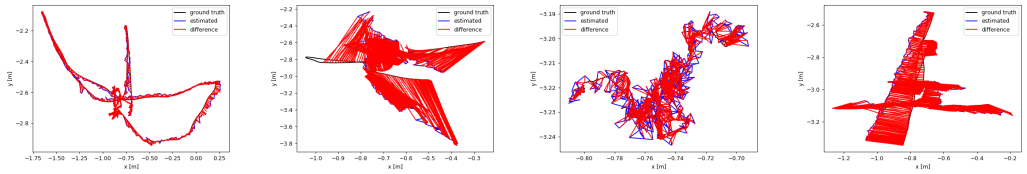
This is used as a constraint to select relatively good data associations (see robust data association algorithm in [13]) and reduce the influence of dynamic objects in tracking for every frame. In the experiment, θ_d is set to 0.6 and θ_s to 0.4.

This module is to estimate the initial camera pose by matching the features between the previous frame and the current frame. Similar to RDS-SLAM, we use the moving probability as the constraint as defined in Eq. (6.14). First, we use features in the static subset. If the matched feature pairs are not enough, we use the features in the unknown subset. If they are still not enough, the features in the dynamic feature subset can also be used such as the ones of a person who is sitting. In the experiment, the dynamic feature subset was not used when evaluating the TUM dataset. The estimated initial pose may not be very reliable; however, it is further optimized via tracking local map and BA.

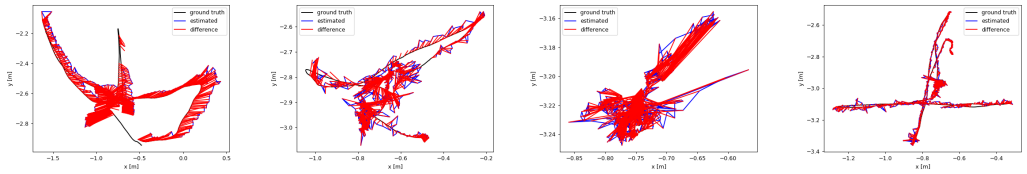
BA is used in the local mapping thread (local BA), the loop closing thread, and the full BA thread. We use moving probability and velocity constraints to filter outliers from them.



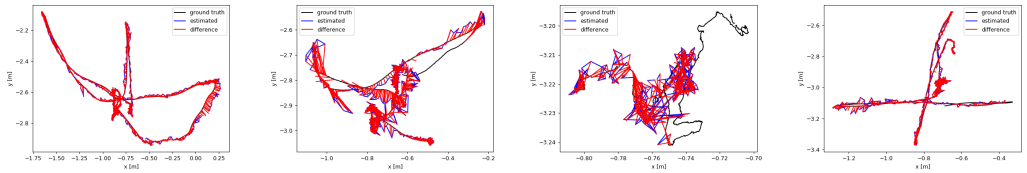
(a) ORB-SLAM3: w/half (b) ORB-SLAM3: w/rpy (c) ORB-SLAM3: w/static (d) ORB-SLAM3: w/xyz



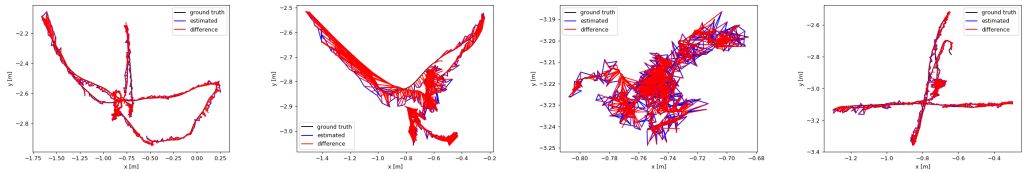
(e) DS-SLAM (S): w/half (f) DS-SLAM (S): w/rpy (g) DS-SLAM (S): w/static (h) DS-SLAM (S): w/xyz



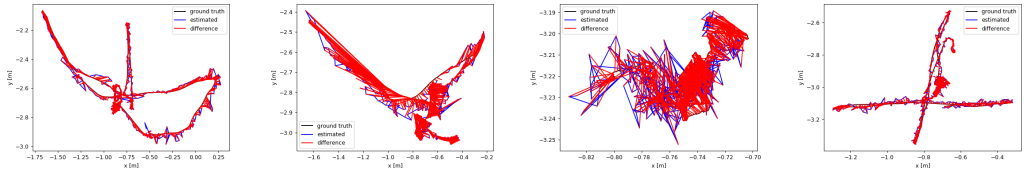
(i) KMOP-vSLAM: w/half (j) KMOP-vSLAM: w/rpy (k) KMOP-vSLAM: w/static (l) KMOP-vSLAM: w/xyz



(m) DynaSLAM (M): w/half (n) DynaSLAM (M): w/rpy (o) DynaSLAM (M): w/static (p) DynaSLAM (M): w/xyz



(q) RDS-SLAM (M): w/half (r) RDS-SLAM (M): w/rpy (s) RDS-SLAM (M): w/static (t) RDS-SLAM (M): w/xyz



(u) RDMO-SLAM (M): w/half (v) RDMO-SLAM (M): w/rpy (w) RDMO-SLAM (M): w/static (x) RDMO-SLAM (M): w/xyz

Figure 6-12: Trajectory comparing frame by brame. "M" stands for "Mask R-CNN" and "S" for "SegNet". RDS-SLAM is executed in 15 Hz and RDMO-SLAM in 30 Hz.

6.6 Experimental results

We demonstrated the real-time performance and the tracking accuracy by comparing with state-of-the-art vSLAMs using the indoor dynamic scenes of the TUM dataset.

Our system was evaluated using GeForce RTX 2080Ti GPU, Cuda 11.1, and an RGB-D camera (Kinect V2). We also showed a demo of AR using a Kinect v2 camera in the real environment.

6.6.1 Tracking Accuracy Evaluation

We compared the trajectories of our proposal with state-of-the-art vSLAM algorithms, as shown in Fig. 6-12, using their source codes when possible, ORB-SLAM3[4], DS-SLAM[5], Dyna-SLAM[6], KMOP-vSLAM [12], and RDS-SLAM[7] using only an RGB-D camera (no IMU).

We evaluated the tracking performance using absolute trajectory error (ATE) and relative pose error (RPE) [49]. The root means squared error (RMSE) and standard deviation (S.D) are used as the error metrics. Given the estimated trajectory: $P_1, ..., P_n \in SE(3)$, ground truth trajectory $Q_1, ..., Q_n \in SE(3)$, and a fixed time interval Δ. The RPE at time i is defined as follows:

$$R_i = (Q^{-1}Q_{i+\Delta})^{-1}(P_i^{-1}P_{i+\Delta}). \tag{6.15}$$

The RMSE of RPE over all time is defined as follows:

$$RMSE(R_{1:n}) = \frac{1}{n}\sum_{\Delta=1}^{n}(\frac{1}{m}\sum_{i=1}^{m}\|trans(R_i)\|^2)^{\frac{1}{2}}. \tag{6.16}$$

The ATE error is defined as follows:

$$A_i = Q_i^{-1}SP_i, \tag{6.17}$$

[4] https://github.com/UZ-SLAMLab/ORB_SLAM3.git
[5] https://github.com/ivipsourcecode/DS-SLAM.git
[6] https://github.com/BertaBescos/DynaSLAM
[7] https://github.com/yubaoliu/RDS-SLAM

where $S \in Sim(3)$, which corresponds to the least squares solution that maps the estimated trajectory onto the ground truth trajectory. The RMSE of ATE over all time indices is defined as follows:

$$RMSE(A_{i:n}, \Delta) = (\frac{1}{n}\sum_{i=1}^{n}\|trans(A_i)\|^2)^{\frac{1}{2}}. \tag{6.18}$$

We compared the tracing performance with counterpart state-of-the-art vSLAMs: ORB-SLAM3 [18], KMOP [12], Detect-SLAM [44], VO-SF [35], Elastic Fusion [40], CO-Fusion [36], Static Fusion [39], DP-SLAM [42], DynaSLAM [41], SLAM-PCD [47], DM-SLAM [46], and RDS-SLAM [13], using, when possible, results published in the original papers, as shown in Tab. 6.1, Tab. 6.2 and Tab. 6.3. We achieved a similar tracking performance with state-of-the-art semantic-based methods in dynamic environments using a heavy segmentation method, Mask R-CNN.

Algorithm 10 Semantic Thread

```
Require: vector<Keyframe*> KF
    Keyframe *requestKF, *workKF, *latestKF
    int workId, latestId, refId, reqId = 0
    int initNum = 1
    thread* segmentThread
 1: while notRequestFinish() do
 2:    if KF.size() < 1 + workId then
 3:       continue
 4:    end if
 5:    workKF = KF[workId]
 6:    latestKF = KF.back()
 7:    latestId = latestKF->id
 8:    if workId < initNum then
 9:       reqKF = workKF
10:       reqId = workId
11:       segment(reqKF)
12:       updateSemantic(reqKF)
13:       refKF = reqKF
14:       refId = reqId
15:       workId++
16:       continue
17:    end if
18:    if (refId >= reqId) || (latestId == reqId) || (workId>=reqId) then
19:       segmentThread->join()
20:    end if
21:    if reqKF->isSemanticReady() && (workId > reqId) && (latestId > reqId)
       then
22:       reqKF = latestKF
23:       reqId = latestId
24:       segmentThread = new thread(&segment, reqKF)
25:    end if
26:    if workKF->isSemanticReady() then
27:       refKF = workKF
28:       refId = workId
29:    else
30:       if (refId<reqId) && (workId>refId) && (workId-refId==1)  then
31:          while !workKF->IsOpticalFlowReady() do
32:             sleep(1)
33:          end while
34:       end if
35:       workKF->label = predictLabel(refKF)
36:       updateSemantic(workKF)
37:       refKF = workKF
38:       refId = workId
39:    end if
40:    workId++
41: end while
```

Algorithm 11 Update Semantic

Require: Keyframe* pKF

```
1: pKF->mask = generateMask(pKF->label)
2: pKF->informSemanticReady()
3: pKF->updateMovingProbability()
```

Table 6.1: Evaluation of absolute trajectory error (ATE) of TUM (m). k-means (K), SegNet (S), Mask R-CNN (M), SharpMask (SM), OpnePose (O) are segmentation or detection methods. RDS-SLAM is evaluated in 15 Hz and ours is evaluated in 30 Hz.

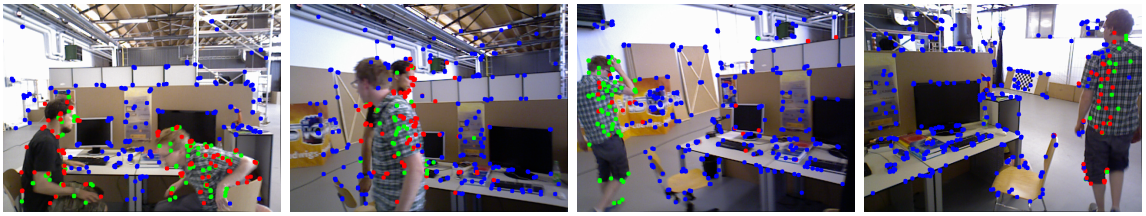
Seq.	ORB SLAM3		KMOP (K+O)	Detect-SLAM (SSD)	VO-SF (K)	Elastic Fusion	CO-Fusion (SM)	Static Fusion (K)	DP-SLAM (M)		DS-SLAM (S)		DynaSLAM (M)		SLAM-PCD (CNN)		DM-SLAM (M)		RDS-SLAM (M)		RDMO-SLAM (M)	
	RMSE	S.D.	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.
w/half	0.6572	0.3124	0.176	0.0514	0.739	0.638	0.803	0.391	0.0254	0.0129	0.0303	0.0159	0.0296	0.0157	**0.0241**	0.0122	0.0274	0.0137	0.0259	0.0141	0.0304	0.0141
w/rpy	1.0197	0.5122	0.049	0.2959	-	-	-	-	**0.0356**	0.0218	0.4442	0.2350	0.0354	0.019	0.0453	0.0316	0.0328	0.0194	0.1468	0.1051	0.1283	0.1047
w/static	0.3614	0.1522	0.032	-	0.327	0.293	0.551	0.014	0.0079	0.0037	0.0081	0.0033	**0.0068**	0.0032	0.0077	0.0039	0.0079	0.0040	0.0815	0.0224	0.0126	0.0071
w/xyz	0.9178	0.4859	0.019	0.0241	0.874	0.906	0.696	0.127	**0.0141**	0.0073	0.0247	0.0161	0.0164	0.0086	0.0157	0.0084	0.0148	0.0072	0.0213	0.0127	0.0226	0.0137
s/static	0.0090	0.0043	-	-	0.029	0.008	0.011	0.013	**0.0059**	0.0029	0.0065	0.0033	0.0108	0.0056	0.0080	0.0037	0.0063	0.0032	0.0088	0.0043	0.0066	0.0033

Table 6.2: Evaluation of translational relative pose error (RPE) (m) of TUM.

Seq.	ORB SLAM3		KMOP (K+O)	VO-SF (K)	Elastic Fusion	CO-Fusion	BaMVO	Static Fusion (K)	DP-SLAM (M)		DS-SLAM (S)		DynaSLAM (M)		SLAM-PCD (CNN)		RDS-SLAM (M)		RDMO-SLAM (M)	
	RMSE	S.D.	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.
w/half	0.3262	0.2625	0.07	0.335	0.205	0.40	0.173	0.207	**0.0142**	0.0082	0.0297	0.0152	0.0284	0.0149	0.0274	0.0140	0.0282	0.0155	0.0294	0.013
w/rpy	0.4368	0.3197	0.065	-	-	-	-	-	**0.0225**	0.0150	0.1503	0.1168	0.0448	0.0262	0.0616	0.0357	0.1114	0.092	0.1396	0.1176
w/static	0.7800	0.7563	0.033	0.101	0.26	0.224	0.133	0.013	**0.0066**	0.0038	0.0102	0.0038	0.0089	0.0044	0.0102	0.0049	0.0419	0.0348	0.016	0.009
w/xyz	0.4258	0.3063	0.026	0.277	0.24	0.329	0.232	0.121	**0.0114**	0.0063	0.0333	0.0229	0.0217	0.0119	0.0204	0.0107	0.0281	0.0167	0.0299	0.0188
s/static	0.0102	0.0049	-	0.024	0.009	**0.0011**	0.024	0.011	0.0054	0.0027	0.0078	0.0038	0.0126	0.0067	0.0087	0.0038	0.0107	0.005	0.009	0.004

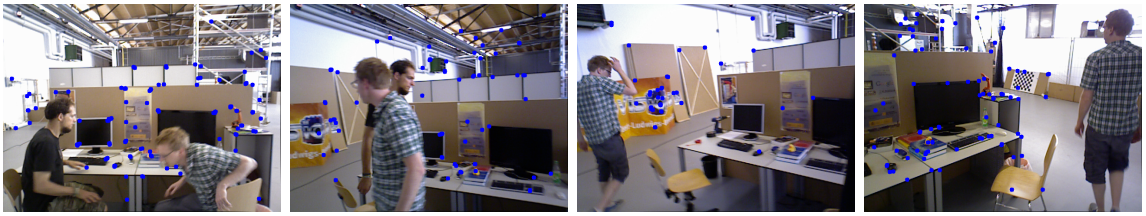
Table 6.3: Evaluation of rotational pose error (RPE) (m) of TUM.

Seq.	ORB SLAM3		KMOP (O+K)	VO-SF (K)	Elastic Fusion	CO-Fusion (SM)	BaMVO	Static Fusion (K)	DP-SLAM (M)		DS-SLAM (S)		DynaSLAM (M)		SLAM-PCD (CNN)		RDS-SLAM (M)		RDMO-SLAM (M)	
	RMSE	S.D.	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.
w/half	7.2352	5.9487	1.595	0.0669	0.0641	0.1302	0.0428	0.0504	**0.0106**	0.0059	0.8142	0.4101	0.7842	0.4012	0.7440	0.3459	0.8216	0.4347	0.7915	0.3782
w/rpy	8.7683	6.4583	1.105	-	-	-	-	-	**0.0128**	0.0082	3.0042	2.3065	0.9894	0.5701	1.3831	0.8318	9.3192	8.572	2.5472	2.0607
w/static	6.0054	5.5995	0.627	0.0168	0.0477	0.0401	**0.0208**	0.0038	0.0044	0.0023	0.2690	0.1215	0.2612	0.1259	0.2631	0.1119	1.1686	0.9917	0.3385	0.1612
w/xyz	7.8974	5.5917	0.689	0.0511	0.0479	0.0555	0.0439	0.0266	**0.0093**	0.0065	0.8266	0.2826	0.6284	0.3848	0.6227	0.3807	0.7236	0.4435	0.799	0.5502
s/static	0.3007	0.1300	-	0.0071	**0.003**	0.0044	0.0069	0.0043	0.0040	0.0021	0.2735	0.1215	0.3416	0.1642	0.2782	0.1210	0.3091	0.1325	0.291	0.133



(a) Frame 2 (b) Frame 50 (c) Frame 100 (d) Frame 205

Figure 6-13: Classify objects according to the moving probability (w/half). Green features are unknown and red ones are dynamic, and blue ones are static.



(a) Frame 2 (b) Frame 50 (c) Frame 100 (d) Frame 205

Figure 6-14: Use robust features in tracking (w/half).

We achieved similar tracking performance compared with the methods that use the blocked model. However, these methods cannot achieve good real-time performance. The proposed method can run the Mask R-CNN version vSLAM in real-time while keeping the robust tracking. We will demonstrate the real-time performance later.

6.6.2 Outlier Removal Using TUM Dataset

We qualitatively checked the feature classification performance by evaluating the TUM dataset. The features can be classified into three subsets according to the moving probability (Eq. (6.14)), as shown in Fig. 6-13. The static features are mostly distributed on static objects, and the unstable features (green and red) are mostly on the moving people. In the tracking thread, we try to use as static features as we can. An example is shown in Fig. 6-14, wherein only selected good static features are used in the initial camera pose estimation stage in the tracking.

(a) Put a virtual cube (b) Before sitting down (c) After sitting down (d) After sitting down

(e) Person left (f) Move the keyboard (g) Disturb the cube (far) (h) Disturb the cube (near)

Figure 6-15: AR demo

6.6.3 AR Demo

We qualitatively evaluated our system using an AR demo, as shown in Fig. 6-15, where a virtual cube is put on the desk. One person is sometimes sitting down and standing up, and sometimes the person occupies half of the camera view. The tracking is very unstable or even tracking lost in the situation such as Figs. 6-15 (b-d) when using the original ORB-SLAM. In this demo, the position of the virtual object is somehow influenced by the person due to the occlusion (e.g., Figs. 6-15 (b) and (d)); however, it recovers to its original position after the person leaves (Fig. 6-15 (e)). We also try to disturb the tracking by moving the keyboard (Fig. 6-15 (f)) and moving the hand (Figs. 6-15 (g) and (h)). Tracking in Figs. 17 (f-h) is not influenced by the hands because features on the hands are detected and removed using semantic and motion information.

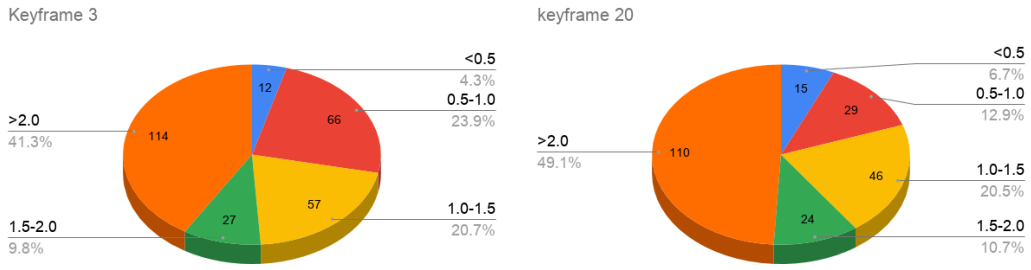
6.6.4 Velocity Constraint vs Semantic Information

We have evaluated the ATE of TUM only using velocity constraint or semantic information, as shown in Tab. 6.4. The tracking performance is much better than that of ORB-SLAM3 with the velocity constraint. This constraint can filter the landmarks (matched with features) that have large velocities on the objects, and it is a little faster than Mask R-CNN segmentation. We also evaluated the tracking performance

Table 6.4: Evaluation of absolute trajectory error (ATE) of TUM (m) with or without velocity and semantic mask. "V" means only use velocity and "M" only use the semantic mask.

Seq.	ORB-SLAM3		RDMO-SLAM (V)		RDMO-SLAM (M)		RDMO-SLAM (V+M)	
	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.	RMSE	S.D.
w/half	7.2352	5.9487	0.6314	0.3518	0.1204	0.0952	**0.0304**	0.0141
w/rpy	8.7683	6.4583	1.0978	0.5768	0.2708	0.2039	**0.1283**	0.1047
w/static	6.0054	5.5995	0.3867	0.1756	**0.0124**	0.0077	0.0126	0.0071
w/xyz	7.8974	5.5917	0.6479	0.3363	**0.0164**	0.0085	0.0226	0.0137
s/static	0.3007	0.1300	0.0089	0.004	**0.0064**	0.003	0.0066	0.0033

that only uses segmentation. The performance may be not good if the camera rotates and translates rapidly because it does not have enough time to obtain semantic information. That is why the tracking performance is a little lower in w/rpy and w/half. This problem can be solved by combining velocity constraints and semantic information. The tracking performances for other scenarios are very similar in the case of only using semantic and using both, especially in the standard deviation.



(a) Keyframe 3 (b) Keyframe 20

Figure 6-16: Landmark distribution according to the velocity range (w/xyz).

6.6.5 Velocity Constraint Threshold

It is challenging to decide the threshold of the velocity to support robust tracking. We analyzed the landmark distribution that matched with features on the keyframes in the terms of the velocity. As shown in Fig. 6-16, the velocity of about a half of landmarks is less than 2.0 in TUM w/xyz. We use the landmarks that have a

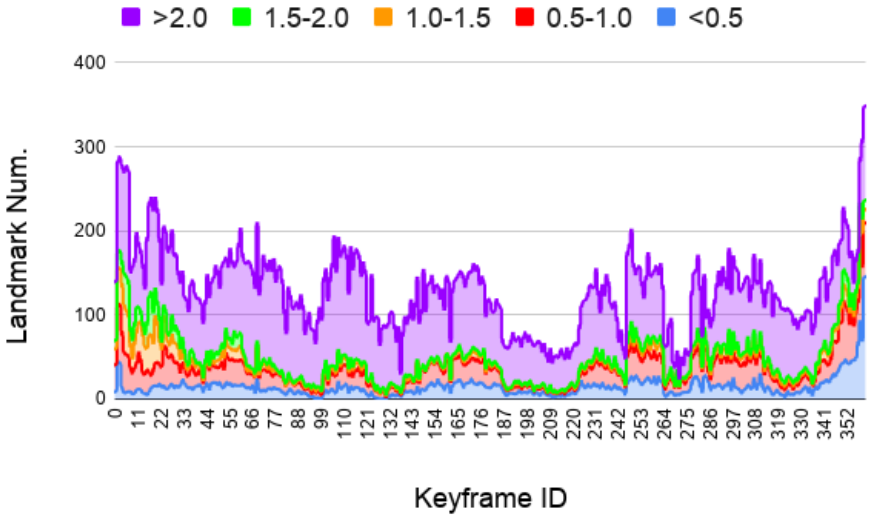


Figure 6-17: The number of landmarks in the different velocity ranges (w/xyz).

relatively small velocity to optimize the camera pose in BA. A very small number of landmarks will be left when setting the threshold too small, and too much noise data are used when setting it too large. We suggest setting the velocity threshold to 1.0-2.0 (see Fig. 6-17 (orange, red, and green lines)) because the number of landmarks used is reasonable in the optimization. To avoid the tracking loss due to the few landmarks, we do not use this constraint in the "track last frame" and "track local map" models in the tracking thread. We only use this constraint in the local BA where many landmarks are used together for optimization.

Table 6.5: The execution time comparison of the TUM Dataset. We use the data in their original paper as possible. If not provided, we approximate the processing time.

Method	Model	GPU	Semantic	Segmentation/Detection Time (ms)	Time of Other Models Related to Tracking (ms)		Total Time for Each Frame (ms)
ORB-SLAM3	-	-	-	-	-	-	22 - 30
KMOP-vSLAM	blocked	GeForce GTX 1080 Ti	Open Pose k-means	45.89 95.29	Geometric Constraints Moving Detection Camera Ego-motion	 221.636	257.819
DP-SLAM	blocked	GeForce MX150	Mask R-CNN	200	Geometric Constraints Background Inpainting Moving probability Updating	- - -	>200
Detect-SLAM	blocked	GTX960M	SSD	310	Progation Updating	20 10	>310
DS-SLAM	blocked	P4000	SegNet	37.57330	ORB feature extraction Moving consistency check	9.375046 29.50869	>65
DynaSLAM	blocked	Tesla M40 GPU	Mask R-CNN	195	Multi-view Geometry Background Inpainting	235.98 (w/rpy) 183.56 (w/rpy)	>300
DM-SLAM	blocked	GeForce GTX 1080 Ti	Mask R-CNN	201.02	Ego-motion Dynamic Point Detection	3.16 40.64	>201
RDS-SLAM (TUM)	non-blocked	GeForce RTX 2080Ti	Mask R-CNN	200	Mask Generation Update Moving Probability Semantic-based Optimization	5.42 0.17 0.54	50 - 65 (15 Hz)
RDMO-SLAM	non-blocked	GeForce RTX 2080Ti	Mask R-CNN	200	Optical flow estimation Update Moving Probability Mask Generation Velocity Estimation Label Prediction	54 0.14 6.04 2.54 1.56	**22-35 (30 Hz)**

6.6.6 Timing Analysis

Tab. 6.5 shows the comparison result of the real-time performance. We compared the time required for the original ORB-SLAM3 (RGB-D camera only), blocked model-based solutions (e.g., DP-SLAM, Detect-SLAM, DS-SLAM, DynaSLAM, DM-SLAM), and non-blocked model-based solutions (e.g., RDS-SLAM). The time required for the blocked model is limited by the time-consuming semantic segmentation, which significantly lowers their real-time performance. Our previous study, RDS-SLAM only can evaluate the TUM dataset at 15 Hz because the TUM dataset is usually short (about half a minute) and Mask R-CNN only can segment very few keyframes, which results in inadequate semantic information when running at 30 Hz. We mitigate this limitation via predicting the semantic label, which enables almost all the keyframes to obtain semantic results even when executing at 30 Hz.

The tracking performance may be influenced by the hardware configuration because the speed of Mask R-CNN and PWC-Net rely on the GPU. However, the time required for tracking each frame is not influenced due to the non-blocked architecture.

6.7 Conclusions

We proposed RDMO-SLAM, a novel real-time vSLAM for the real environment exploiting RDS-SLAM, Mask R-CNN, and dense optical flow. To overcome the problem of inadequate semantic information obtained within a short time due to the slow speed of Mask R-CNN segmentation, we predict semantic labels using optical flow so that almost all the keyframes can acquire the semantic information. To reduce the influence of dynamic objects untrained by semantic segmentation models, we add a velocity constraint by estimating the velocity of landmarks using optical flow. The tracking and real-time performances are evaluated using the dynamic scenes of the TUM RGB-D dataset and compared with counterpart state-of-the-art vSLAMs with similar motivation. As a result, our proposal that uses a non-blocked model can maintain real-time nature (30 Hz) even with a very heavy segmentation method. In future works, we will 1) consider the outdoor environment and 2) build a static dense

map without dynamic objects.

Chapter 7

Conclusions

7.1 Summary

Visual simultaneous localization and mapping (vSLAM) is considered a fundamental technology for augmented reality and intelligent mobile robots. However, rigid scene assumption is common in vSLAM, which limits the wide usage in populated real-world environments. To deal with the challenge, geometric-based, reconstruction-based, and semantic-based solutions are proposed. It is a great challenge to balance tracking performance and real-time performance. The state-of-the-art proposals cannot trade off them properly. Pure geometric-based solutions usually cannot achieve good tracking performance without the aid of semantic information. Some reconstruction-based algorithms need a 3D model, which is not suitable for real-time performance and wide deployment in the real environment. To our best knowledge, the semantic-based algorithms use the blocked model that limits the real-time performance.

To apply the system in a robot where GPU may not be deployed and compatible with many semantic segmentation methods, we proposed RTS-vSLAM. The tracking accuracy is greatly improved comparing to the original ORB-SLAM2. Semantic segmentation can only deal with pre-defined objects. Therefore, in KMOP-vSLAM, we try to use k-means to segments all the objects and use OpenPose to detect people. The tracking accuracy is not better than the semantic-based algorithms and it is difficult to decide the optimal k value of k-means. Both RTS-vSLAM and KMOP-

vSLAM cannot achieve good real-time tracking performance because they use the blocked model, that is, they need to wait for the semantic result before tracking.

To keep the real-time nature while maintaining the robust tracking, we proposed RDS-SALAM that uses the non-blocked model, that is, the tracking process is not blocked by waiting for the semantic result. In RDS-SAM, the semantic thread that uses SegNet and Mask R-CNN runs in parallel with the tracking thread. The SegNet version can run in 30HZ with robust tracking accuracy, however, the Mask R-CNN version needs to run in 15HZ to maintain the robust tracking accuracy because Mask R-CNN is slow, and not enough semantic information can be obtained in a short time. To obtain the semantic information more quickly and evaluate vSLAM in 30HZ even using a heavy segmentation method like Mask R-CNN, we proposed RDMO-SLAM that uses a novel semantic label prediction algorithm with the aid of dense optical flow. Considering the semantic segmentation can only deal with the pre-defined objects, we estimate the velocity of each features using the optical flow as an additional constraint to reduce the influence of the undefined objects.

7.2 Future work

There are some remaining problems. First, the semantic segmentation methods (e.g., Mask R-CNN) usually are time-consuming. It is better to customize the neural network to faster the prediction. Second, we need to obtain some information using semantic methods, such as the bounding box, semantic label, optical flow, the depth information of the image, and the key points of objects. This information is beneficial to improve localization accuracy and build static semantic maps. It is better to generate them using a unified neural network or a multi-task neural network. Third, the RGB-D camera is not suitable for outdoor environments. It is better to extend our algorithms to outdoor scenarios using a mono camera or stereo camera. Fourth, multiple sensors can be fused to get more robust tracking accuracy using Lidar, IMU, and GPS.

In future work, we will try to 1) deploy our system on a real robot, 2) extend

our system to the stereo camera and mono camera systems, and 3) merge neural networks, e.g., semantic segmentation and optical flow into one unified framework.

Bibliography

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robotics and Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[2] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, 2017.

[3] R. Mur-Artal, J. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[4] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard, "3D Mapping with an RGB-D camera," *IEEE Transactions on Robotics*, vol. 30, no. 1, pp. 177–187, 2014.

[5] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct Monocular SLAM," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 834–849, 2014.

[6] J. Engel, V. Koltun, and D. Cremers, "Direct Sparse Odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 3, pp. 611–625, 2018.

[7] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[8] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2O: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, IEEE, 2011.

[9] J. Zhang, M. Henein, R. Mahony, and V. Ila, "VDO-SLAM: A Visual Dynamic Object-aware SLAM System," *Journal of Vibration and Control*, 2020.

[10] B. Bescos, C. Campos, J. D. Tardos, and J. Neira, "DynaSLAM II: Tightly-Coupled Multi-Object Tracking and SLAM," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5191–5198, 2021.

[11] Y. Liu and J. Miura, "RTS-vSLAM : Real-time Visual Semantic Tracking and Mapping under Dynamic Environments," in *16th International Symposium on Intelligent Autonomous Systems (IAS-16)*, pp. 1–12, 2021.

[12] Y. Liu and J. Miura, "KMOP-vSLAM: Dynamic Visual SLAM for RGB-D Cameras using K-means and OpenPose," in *2021 IEEE/SICE International Symposium on System Integration (SII)*, (Iwaki, Japan), pp. 415–420, IEEE, 2021.

[13] Y. Liu and J. Miura, "RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods," *IEEE Access*, vol. 9, pp. 23772–23785, 2021.

[14] Y. Liu and J. Miura, "RDMO-SLAM: Real-Time Visual SLAM for Dynamic Environments Using Semantic Label Prediction With Optical Flow," *IEEE Access*, vol. 9, pp. 106981–106997, 2021.

[15] G. Klein and D. Murray, "Parallel tracking and mapping on a camera phone," in *Science and Technology Proceedings - IEEE 2009 International Symposium on Mixed and Augmented Reality, ISMAR 2009*, pp. 83–86, 2009.

[16] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2564–2571, 2011.

[17] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, 2017.

[18] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM," *IEEE Transactions on Robotics*, pp. 1–17, 2021.

[19] R. Elvira, J. D. Tardos, and J. M. Montiel, "ORBSLAM-Atlas: A robust and accurate multi-map system," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 6253–6259, 2019.

[20] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, 2017.

[21] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," *IEEE International Conference on Intelligent Robots and Systems*, pp. 2100–2106, 2013.

[22] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[23] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[24] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2980–2988, 2017.

[25] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[26] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv:1804.02767*, 2018.

[27] D. Bolya, C. Zhou, F. Xiao, and Y. J. Lee, "YOLACT: Real-time instance segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 9156–9165, 2019.

[28] Z. Cao, T. Simon, S. E. Wei, and Y. Sheikh, "Realtime multi-person 2D pose estimation using part affinity fields," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, pp. 1302–1310, 2017.

[29] S. Li and D. Lee, "RGB-D SLAM in Dynamic Environments Using Static Point Weighting," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2263–2270, 2017.

[30] Y. Sun, M. Liu, and M. Q.-H. Meng, "Motion removal for reliable RGB-D SLAM in dynamic environments," *Robotics and Autonomous Systems*, vol. 108, pp. 115–128, 2018.

[31] J. Cheng, Y. Sun, and M. Q. Meng, "Improving monocular visual SLAM in dynamic environments: an optical-flow-based approach," *Advanced Robotics*, vol. 33, no. 12, pp. 576–589, 2019.

[32] D. H. Kim, S. B. Han, and J. H. Kim, "Visual odometry algorithm using an RGB-D sensor and IMU in a highly dynamic environment," in *Advances in Intelligent Systems and Computing*, vol. 345, pp. 11–26, Springer Verlag, 2015.

[33] Y. Sun, M. Liu, and M. Q. Meng, "Improving RGB-D SLAM in dynamic environments: A motion removal approach," *Robotics and Autonomous Systems*, vol. 89, pp. 110–122, 2017.

[34] W. Tan, H. Liu, Z. Dong, G. Zhang, and H. Bao, "Robust monocular SLAM in dynamic environments," in *2013 IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2013*, pp. 209–218, IEEE, 2013.

[35] M. Jaimez, C. Kerl, J. Gonzalez-Jimenez, and D. Cremers, "Fast odometry and scene flow from RGB-D cameras based on geometric clustering," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3992–3999, IEEE, May 2017.

[36] M. Runz and L. Agapito, "Co-fusion: Real-time segmentation, tracking and fusion of multiple objects," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 4471–4478, IEEE, 2017.

[37] P. O. Pinheiro, T. Y. Lin, R. Collobert, and P. Dollár, "Learning to refine object segments," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9905 LNCS, pp. 75–91, 2016.

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

[39] R. Scona, M. Jaimez, Y. R. Petillot, M. Fallon, and D. Cremers, "StaticFusion: Background Reconstruction for Dense RGB-D SLAM in Dynamic Environments," in *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3849–3856, IEEE, 2018.

[40] T. Whelan, S. Leutenegger, R. Salas Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM Without A Pose Graph," in *Robotics: Science and Systems XI*, vol. 11, Robotics: Science and Systems Foundation, 2015.

[41] B. Bescos, J. M. Facil, J. Civera, and J. Neira, "DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4076–4083, 2018.

[42] A. Li, J. Wang, M. Xu, and Z. Chen, "DP-SLAM: A visual SLAM with moving probability towards dynamic environments," *Information Sciences*, vol. 556, pp. 128–142, 2021.

[43] C. Yu, Z. Liu, X. J. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei, "DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 1168–1174, IEEE, 2018.

[44] F. Zhong, S. Wang, Z. Zhang, C. Chen, and Y. Wang, "Detect-SLAM: Making Object Detection and SLAM Mutually Beneficial," in *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018*, pp. 1001–1010, 2018.

[45] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *Lecture Notes in Computer Science*, vol. 9905, pp. 21–37, Springer, Cham, 2016.

[46] J. Cheng, Z. Wang, H. Zhou, L. Li, and J. Yao, "DM-SLAM: A feature-based SLAM system for rigid dynamic scenes," *ISPRS International Journal of Geo-Information*, vol. 9, no. 4, pp. 1–18, 2020.

[47] Y. Fan, Q. Zhang, S. Liu, Y. Tang, X. Jing, J. Yao, and H. Han, "Semantic SLAM With More Accurate Point Cloud Map in Dynamic Environments," *IEEE Access*, vol. 8, pp. 112237–112252, 2020.

[48] N. Dvornik, K. Shmelkov, J. Mairal, and C. Schmid, "BlitzNet: A Real-Time Deep Network for Scene Understanding," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 4174–4182, 2017.

[49] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *IEEE International Conference on Intelligent Robots and Systems*, pp. 573–580, 2012.

[50] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

[51] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Ur-

ban Scene Understanding," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3213–3223, IEEE, Apr 2016.

[52] S. Song, S. P. Lichtenberg, and J. Xiao, "SUN RGB-D: A RGB-D scene understanding benchmark suite," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 567–576, IEEE, 2015.

[53] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene Parsing through ADE20K Dataset," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5122–5130, IEEE, 2017.

[54] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[55] Y. Wang and S. Huang, "Motion segmentation based robust RGB-D SLAM," in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, no. March, pp. 3122–3127, IEEE, Jun 2014.

[56] W. Dai, Y. Zhang, P. Li, and Z. Fang, "RGB-D SLAM in Dynamic Environments Using Points Correlations," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2263–2270, 2018.

[57] R. Wang, W. Wan, Y. Wang, and K. Di, "A new RGB-D SLAM method with moving object detection for dynamic indoor scenes," *Remote Sensing*, vol. 11, no. 10, 2019.

[58] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2nd ed., 2003.

[59] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1027–1035, 2007.

[60] L. Xiao, J. Wang, X. Qiu, Z. Rong, and X. Zou, "Dynamic-SLAM: Semantic monocular visual localization and mapping based on deep learning in dynamic environment," *Robotics and Autonomous Systems*, vol. 117, pp. 1–16, 2019.

[61] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Lecture Notes in Computer Science*, vol. 8693 LNCS, pp. 740–755, Springer, Cham, 2014.

[62] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2005.

[63] D. Sun, X. Yang, M.-Y. Liu, and J. Kautz, "PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume," in *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8934–8943, IEEE, 2018.

[64] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to digit recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[65] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1647–1655, IEEE, 2017.

[66] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A Naturalistic Open Source Movie for Optical Flow Evaluation," in *Lecture Notes in Computer Science*, pp. 611–625, Springer-Verlag, 2012.

List of Publications

Journal Articles

[1] Y. Liu and J. Miura, RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods, IEEE Access, vol. 9, pp. 23772–23785, Jan. 2021, doi: 10.1109/ACCESS.2021.3050617.

[2] Y. Liu and J. Miura, RDMO-SLAM: Real-Time Visual SLAM for Dynamic Environments Using Semantic Label Prediction With Optical Flow, IEEE Access, vol. 9, pp. 106981–106997, 2021, doi: 10.1109/ACCESS.2021.3100426.

International Conference Papers

[1] Y. Liu and J. Miura, KMOP-vSLAM: Dynamic Visual SLAM for RGB-D Cameras using K-means and OpenPose, in 2021 IEEE/SICE International Symposium on System Integration (SII), pp. 415-420, 2021, doi: 10.1109/IEEECONF49454.2021.9382724.

[2] Y. Liu and J. Miura, RTS-vSLAM : Real-time Visual Semantic Tracking and Mapping under Dynamic Environments, in 16th International Symposium on Intelligent Autonomous Systems (IAS-16), pp. 1–12, 2021.

Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Jun Miura for the continuous support of my Ph.D. study, for his expertise, understanding, and patience. I would like to thank him for his assistance in writing journals, reports, also for giving me the chance to attend prestigious conferences. I appreciate all his advice on not only research activities but also on my career plan, which had a great influence on my life. Besides my supervisor, I would also like to express my gratitude to Prof. Kuriyama and Prof. Sugaya as the examiner members who kindly provide many fruitful suggestions on this thesis.

I thank all my lab-mates at the Advanced Intelligent Systems Laboratory (AISL): Kenji Koide, Chandra Kusuma Dewa, Yasunori Kawamata, Liliana Villamar Gomez, Shigemichi Matsuzaki, Kazuki Mano, Mahmood UL Hassan, Oskar Natan, Shunsuke Ochi, Yusuke Miake, Hoai Luu Duc, Masataka Inouchi, Keishi Ishihara, Hafiza Ufaq Rehman, Ziyad Tareq Nouri, Sho Tatsuguchi, Dipankar Das, Sinem Gozde Defterli, Amadou Cisse, Hiroaki Masuzawa, Kotaro Hayashi, and Hironori Fujimoto, for not only their cooperation on my work but also all the fun we have had in the lab.

I wish to thank the Japanese Ministry of Education, Culture, Sports, Science, and Technology (MEXT) scholarship that has provided generous financial support for my study and living expenses.