SOLO-SLAM: A Parallel Semantic SLAM Algorithm for Dynamic Scenes

Liuxin Sun, Junyu Wei *, Shaojing Su and Peng Wu

Abstract: Simultaneous localization and mapping (SLAM) is a core technology for mobile robots working in unknown environments. Most existing SLAM techniques can achieve good localization accuracy in static scenes, as they are designed based on the assumption that unknown scenes are rigid. However, real-world environments are dynamic, resulting in poor performance of SLAM algorithms. Thus, to optimize the performance of SLAM techniques, we propose a new parallel processing system, named SOLO-SLAM, based on the existing ORB-SLAM3 algorithm. By improving the semantic threads and designing a new dynamic point filtering strategy, SOLO-SLAM completes the tasks of semantic and SLAM threads in parallel, thereby effectively improving the real-time performance of SLAM systems. Additionally, we further enhance the filtering effect for dynamic points using a combination of regional dynamic degree and geometric constraints. The designed system adds a new semantic constraint based on semantic attributes of map points, which solves, to some extent, the problem of fewer optimization constraints caused by dynamic information filtering. Using the publicly available TUM dataset, SOLO-SLAM is compared with other state-of-the-art schemes. Our algorithm outperforms ORB-SLAM3 in accuracy (maximum improvement is 97.16%) and achieves better results than Dyna-SLAM with respect to time efficiency (maximum improvement is 90.07%).

Keywords: SLAM; SOLO-SLAM; SOLO_V2; deep learning; navigation; robotics

1. Introduction

Simultaneous localization and mapping (SLAM) is one of the core technologies in the field of robotics. Based on slam technology, mobile robots can realize their real-time localization and environment map construction through sensor data in unknown environments. Owing to the wide range of promising applications in autonomous driving, robotics, and virtual reality technologies, visual SLAM techniques have received increasing attention from researchers in recent years. After long-term research, the current visual slam system has formed a basic framework composed of visual odometry, state estimation backend, map construction, and loopback detection [1]. Based on the above classical framework, several SLAM algorithms have been proposed, such as ORB-SLAM3 [2], VINS-Fusion [3], DSO [4], etc. Several of these algorithms have excellent performance and can already meet the application requirements in most cases. The theoretical framework of the SLAM system has become relatively complete. However, as the application scenarios become increasingly complex, the SLAM system also shows obvious shortcomings in some specific real-world scenarios. For example, when the robot runs in an indoor scene, it will inevitably face the situation of people walking. In the classical SLAM framework, both the feature point method and the direct method rely on the strong assumptions of rigid scenarios. This assumption limits the application of SLAM algorithms in real-world environments. The features extracted by the SLAM system from moving objects (such as pedestrians) will lead to incorrect data association results. This will greatly interfere with the visual localization and map building processes [5].

In order to solve this problem, researchers have gradually formed three improvement directions [6]: (1). a filtering method based on the conventional model. Researchers filter...
outliers based on traditional methods such as RANSAC, optical flow [7], and epipolar constraints. (2). A filtering method based on the deep learning model. The SLAM system is based on the use of a deep learning segmentation algorithm to obtain the a priori dynamic regions in the image. After that, the feature points within the a priori dynamic region are filtered out. (3). An enhancement method based on the deep learning model. Researchers further enhance the SLAM system based on the semantic information obtained by the second method. Since the deep learning-based method usually has a better filtering effect, the system can usually achieve better localization accuracy. In contrast, the segmentation task of deep learning will consume a great deal of time. Taking Dyna-SLAM [8], a classical open-source solution based on the second improvement idea, as an example, the system needs to complete the segmentation task to get the semantic region before it can further perform the bit pose estimation in the tracking thread of the SLAM system. This also leads to the overall processing time being the sum of the time consumed by the segmentation task and the processing time of the SLAM system, and the overall real-time performance is poor.

Aiming at the problem of positioning accuracy and real-time operation in indoor dynamic scenes, this paper proposes a new SLAM algorithm (SOLO-SLAM), which combines the above three improvement ideas. The algorithm performs two rounds of dynamic point filtering design by combining instance segmentation and geometric constraints. This greatly enhances the localization accuracy of the system in dynamic scenes. Distinct from algorithms such as Dyna-SLAM, SOLO-SLAM will update the dynamic state and semantic attributes of mapped points in a separate semantic thread. By design, SOLO-SLAM can perform filtering operations without waiting for the semantic segmentation results of the current frame. This greatly enhances the real-time performance of the system. In addition, SOLO-SLAM also designs a new constraint based on the semantic information of the environment and the semantic attributes of map points to improve the constraint reduction problem caused by dynamic point filtering. The main contributions of our approach are summarized as follows:

1. We design a SLAM system (SOLO-SLAM) in which the segmentation and pose estimation tasks are fully processed in parallel. SOLO-SLAM updates the dynamic state and semantic attributes of map points in the semantic thread. When processing the current frame, the system can perform the first round of filtering according to the dynamic state of map points without waiting for the semantic segmentation result of the current frame.

2. We propose an auxiliary filtering method based on geometric constraints combined with the dynamic ranking of regions. The method performs different levels of filtering strategies for different regions based on geometric constraints by dividing the dynamic degree of the regions.

3. We introduce a new semantic constraint based on the semantic attributes of map points. The method performs a new constraint construction based on the relationship between the semantic attributes of map points and the semantic region of the current frame.

The remainder of this paper is structured as follows. Section 2 describes related work. Section 3 outlines our proposed algorithm. Section 4 presents the experimental results in detail. Section 5 discusses our results and summarizes the main conclusions.

2. Related Works

At present, the basic design idea of the SLAM system for dynamic scenes is to filter the information introduced by moving objects as outliers. Based on this design idea, the general work can be divided into (1) filtering methods based on the traditional model and (2) filtering methods based on the deep learning model. Our proposed SOLO-SLAM system adopts a two-round filtering strategy. In the first round of filtering, it is based on the deep learning method. In the second round of filtering, it is based on the traditional geometric model approach. We present the current relevant excellent work in Sections 2.1 and 2.2, respectively.

With the development of deep learning, researchers are increasingly trying to use the semantic information of the environment obtained from deep learning to enhance the
performance of the system. In the filtering process of the SOLO-SLAM system, we can get the semantic information of the environment. Therefore, we tried to design a new semantic constraint to enhance the performance of the system by using the semantic information. We introduce relevant work in Section 2.3.

2.1. Filtering Method Based on Conventional Model

Before the widespread use of deep learning, researchers used traditional models for filtering. The related work is as follows. Kim et al. [9] proposed the background model-based dense-visual-odometry (BaMVO) algorithm for RGB-D sensors. This algorithm computes a nonparametric background model to represent the background information in the scene based on the acquired scene depth. Then, the anomalous motion information in the scene is extracted and filtered by means of background-based subtraction. Sun et al. [10] used particle filtering and applied the maximum a posteriori (MAP) method for vector quantized images to enhance the boundary information of moving objects obtained based on the temporal difference technique. However, the computational overload of the algorithm and the requirement that the static region must be the main region somewhat limit the practical value of the algorithm. Li et al. [11] assigned static weights to the depth edge points obtained based on depth discontinuities. They proposed a static weighting method for depth edge points in key frames to assist in reducing the impact caused by dynamic information. Mur-Artal et al. [12] utilized the classical outlier filtering algorithm RANSAC [13] in ORB-SLAM2 to filter the more representative feature points as much as possible, thus reducing the impact caused by anomalous data. Pire et al. [14] used a robust cost function in PTAM to reduce the impact of introduced outliers on pose optimization.

2.2. Filtering Method Based on Deep Learning Model

With the development of deep learning (DL), numerous breakthroughs have been achieved with respect to object detection and segmentation, especially for dynamic information filtering. DL-based segmentation algorithms can identify a priori dynamic regions in an image. After that, the feature points within the a priori dynamic regions are filtered. Compared to conventional models, depth methods can usually filter out dynamic information better. Therefore, systems based on depth models can usually achieve better localization accuracy.

To achieve precise localization and map building, B. Bescos et al. [8] proposed DynaSLAM based on ORB-SLAM2, with the incorporation of dynamic object detection. The filtering of dynamic objects involves two processes: obtaining semantic segmentation results for dynamic regions in the scene based on the MaskRcnn algorithm [15] and implementing the region growth algorithm to grow regions around dynamic pixels in the image. By combining the two strategies, Dyna-SLAM achieves almost complete filtering of the key points in dynamic regions. Nevertheless, Dyna-SLAM, a classical open-source DL-based approach, needs to complete the segmentation task of identifying the semantic region before it can further perform the task of pose estimation and tracking, leading to poor real-time performance. Yu et al. [16] proposed DS-SLAM by introducing a semantic segmentation approach that combines a semantic segmentation network with a moving consistency check method to reduce the influence of dynamic objects, thus improving localization accuracy in dynamic environments. Additionally, DS-SLAM can generate dense semantic octree graphs for advanced tasks such as navigation. Furthermore, PSP-SLAM, proposed by Long et al. [17], incorporates the idea of semantic segmentation. By utilizing the reverse ant colony strategy, the system avoids dynamic and static judgments for all feature points, reducing processing time during detection. All above-mentioned DL-based algorithms have achieved better localization accuracy compared to the traditional methods. However, they show poor real-time performance due to the substantial amount of time consumed by DL methods. In their Detect-SLAM framework, Zhong et al. [18] employed a DL-based algorithm (i.e., single shot multi-box object detector (SSD)) to detect a priori dynamic objects in the environment. When SSD detects an a priori dynamic object, it removes all feature
points in the object region and simultaneously updates the dynamic information of the map points. Subsequently, the system uses dynamic information to assist in feature point selection. Although Detect-SLAM is also based on the sequential processing of detecting a priori regions and then filtering them, it updates the dynamic information of map points to assist in feature selection. We were partly inspired by this in our work. Similarly, Liu et al. [19] developed RDS-SLAM, in which the dynamic probability of map points is updated based on a semantic segmentation algorithm to assist the system in dynamic point filtering. However, both of these schemes have certain problems that are illustrated in more detail in the next section. We also verify the effectiveness of our algorithm by comparing its performance with that of Detect-SLAM and RDS-SLAM in Section 4.

2.3. Enhancement Method Based on Deep Learning Model

Several studies have explored the utilization of obtained environmental information (e.g., semantic information) to improve system performance. E. Sucar et al. [20] proposed a monocular scale correction algorithm based on semantic information. This algorithm first acquires the boundaries of vehicles based on a target detection algorithm (YOLO). Then, the actual height of the vehicle is obtained by projecting the 3D points in the local map onto a 2D image to determine the 3D points corresponding to the boundary points of the vehicle area. Finally, scale correction is performed based on the actual and a priori height of the vehicle. Zhang et al. [21] used PWC-Net [22] to obtain the corresponding optical flow residuals of RGB images to highlight the dynamic regions in RGBD point clouds, thereby enabling self-motion estimation and static semantic background reconstruction excluding dynamic information. To enhance optimization, Lianos et al. [23] constructed a new cost function based on the distance relationship between feature points with vehicle semantic attributes and the vehicle mask area in the image. Our semantic optimization improvements are inspired by related ideas. Based on this, we propose a semantic optimization approach supporting up to 80 semantic categories for indoor scenes. Simultaneously, we adopt a new idea for constructing cost functions. Additionally, our constraint construction approach is more reasonable compared to VSO.

3. Materials and Methods

In this section, we introduce SOLO-SLAM from three aspects. First, we outline the overall framework of SOLO-SLAM. The main contents of the overall framework are 1. the improvements over ORB-SLAM3 and 2. the advantages of improving the real-time performance of the system compared to common algorithms incorporating semantic segmentation. Afterwards, we introduce the filtering algorithm based on the dynamic degree of the region. This method combines semantic information and geometric constraints to filter anomalous data introduced by dynamic map points. Finally, we develop an optimization strategy based on semantic information of indoor scenes. The method is implemented based on the semantic attributes of map points and the semantic regions on pictures obtained from instance segmentation.

3.1. Overview of SOLO-SLAM System

The SOLO-SLAM system aims to improve the performance of ORB-SLAM3. It includes five threads to process tasks. As shown in Figure 1, ORB-SLAM3 consists of three main threads: tracking, local mapping, and loop closing. Based on these threads, ORB-SLAM3 shows excellent performance in static scenarios. For the necessity of program implementation, we improved the orange part in Figure 1. The storage of instance segmentation results was added to the KeyFrame. Semantic attributes, dynamic states, and modifications related to nearest neighboring points were added to MapPoints. The track local map module added new support for dynamic information processing. Specific applications are explained step by step with the new contents below.
SLAM3 shows excellent performance in static scenarios. For the necessity of program implementation, we improved the orange part in Figure 1. The storage of instance segmentation results was added to the KeyFrame. Semantic attributes, dynamic states, and modifications related to nearest neighboring points were added to MapPoints. The track local map module added new support for dynamic information processing. Specific applications are explained step by step with the new contents below.

Figure 1. Overall structure of the SOLO-SLAM system (blue area is the new content added compared to ORB-SLAM3; the dashed box offers detailed description of the module; and the orange area presents the content modified).

We added three main parts on top of the above. As shown in the blue part of Figure 1, we first added a new semantic thread. This thread operates on key frames. Its functions are as follows.

The semantic thread is responsible for saving the newly created key frames and sending segmentation requests to the SOLO-V2 [24] instance segmentation function package based on the Robot Operating System (ROS). After obtaining the instance segmentation results, the dynamic probability and semantic attributes of map points in the system are updated in conjunction with the mask of the key frames. We describe this in detail from a real-time perspective in Section 3.2.

Inside the original tracking thread of ORB-SLAM3, we added a new filtering module for dynamic features. The details are as follows.

Our overall design of the dynamic feature filtering module is shown in Figure 2. The module is divided into two parts: (a) and (b). Figure 2a illustrates the preliminary filtering method based on the dynamic probability of map points. The basic idea is to update the dynamic probability of map points in the semantic thread based on the segmentation result of the a priori dynamic region. Consequently, the dynamic state of the map point is updated based on the dynamic probability. Finally, all the projection points corresponding to the dynamic map points are filtered in the tracking thread. It can be seen that there are still some blue points in the region corresponding to the human. Therefore, we perform a second round of filtering. As shown in Figure 2b, the operation can be divided into three steps: 1. computing the dynamic degree of the region, 2. filtering using an epipolar constraint, and 3. re-judgment based on the depth error. We introduce the method for the second round of filtering in detail in Section 3.3.
constraint, and 3. re-judgment based on the depth error. We introduce the method for the second round of filtering in detail in Section 3.3.

Figure 2. The dynamic point filtering method of SOLO-SLAM. (a) corresponds to the first round of filtering. (b) corresponds to the second round of filtering. The red points represent the map point projection identified as dynamic points in the first round of filtering. The blue points represent the map point projection identified as static points in the first round of filtering. The green points selected in the yellow rectangular box correspond to the points filtered in the second round.

In addition, we added a semantic optimization thread for key frame operations, with the following function.

The semantic optimization thread is responsible for saving the key frames processed by the semantic thread. Additionally, it is responsible for building new semantic constraints based on the relationship between semantic attributes and semantic masks of map points. The semantic optimization method is combined with the optimization method based on the re-projection error. Relevant implementation details are described in Section 3.4.

The above new semantic threads and semantic optimization threads run completely in parallel with other threads, thus fully guaranteeing the independence of the tracking threads. We describe the above three improvements below.
3.2. Real-Time Oriented Dynamic Information Filtering Method

3.2.1. Introduction to the Parallel Processing Design

Usually, after training DL models, we can obtain the location of a priori dynamic targets in the scene based on semantic segmentation methods. Therefore, in recent years, researchers have developed numerous DL-based methods for identifying dynamic target regions and consequently filtering dynamic features in visual odometry. However, these methods suffer from serious real-time problems. Figure 3a shows that in most existing methods, dynamic region filtering is added to the tracking thread. Then, an additional semantic segmentation requests in the semantic thread into the tracking thread, it will not have too much impact on the whole system time. In this article, we call this method sequential processing. In contrast, our proposed SOLO-SLAM enables parallel processing of the instance segmentation task and the pose calculation task.

Among the three, the semantic segmentation time is usually much longer than the feature extraction time. Usually, after training DL models, we can obtain the location of a priori dynamic targets in the scene. Therefore, in this approach, the semantic thread only plays the role of sending semantic segmentation requests, while the tracking thread must wait for the segmentation task to be completed before filtering the dynamic feature points. Therefore, although the above method adds a semantic thread, in fact, the calculation of pose still needs to wait for the completion of the segmentation task.

Since the semantic segmentation method takes a long time, whereas feature extraction often takes very little time, the actual time consumed by the tracking thread is the sum of the semantic segmentation time \( t_{sem} \), the dynamic feature filtering processing time \( t_{process} \), and the pose estimation time \( t_{track} \).

Among the three, the semantic segmentation time is usually much longer than the sum of the latter two. Therefore, to a certain extent, even if we integrate the task of sending semantic segmentation requests in the semantic thread into the tracking thread, it will not have too much impact on the whole system time. In this article, we call this method sequential processing. In contrast, our proposed SOLO-SLAM enables parallel processing of the instance segmentation task and the pose calculation task.
Our proposed SOLO-SLAM is based on the motion model for tracking after initialization is completed. The motion model [2] is implemented as follows:

Step 1: The pose transformation result obtained from the previous frame of the system is used as the initial value.

Step 2: Project map points to the current frame for matching.

Step 3: According to the matching results, graph optimization is performed based on the re-projection error to obtain the pose of the current frame.

As shown in Figure 2a, sequential processing filters all the projection points in the a priori dynamic region based on the obtained mask. When tracking based on the motion model, our projection points are derived from the projection of map points. In essence, filtering the projection points in the a priori dynamic target mask can be transformed into identifying the dynamic map points. When the map points are dynamic, filtering can be achieved without using the corresponding projection points.

Therefore, we specifically design the parallel processing method shown in Figure 2b. Compared to the sequential processing approach, the pose calculation for the current frame can be performed directly without waiting for the instance segmentation task. The workflow is as follows:

Step 1: Filter the feature points matched by the dynamic map points in the current frame in the tracking thread.

Step 2: Calculate the pose based on the filtered feature points in the tracking thread.

The difference between our method and the sequential processing method illustrated in Figure 1 is that we do not need the semantic segmentation results for each frame. Therefore, the SLAM system can reduce the semantic segmentation time \( t_{\text{sem}} \) in the tracking thread.

\[
\begin{align*}
  t_{\text{total_sp}} &= t_{\text{sem}} + t_{\text{process}} + t_{\text{track}} \\
  t_{\text{total_pp}} &= t_{\text{process}} + t_{\text{track}}
\end{align*}
\]  

(1)

In the above parallel processing steps, when it is determined that the map points are dynamic points, the corresponding projection matching results are filtered. The dynamic state of map points is determined by the dynamic probability of map points. Therefore, the problem is further transformed into the updating of the map point probability in the semantic thread.

3.2.2. Dynamic State Update Implementation of Map Points

As shown in Algorithm 1, we show the specific process of the dynamic probability update in the semantic thread. Next, we present the details of the dynamic probability update and dynamic status update of map points.

**Dynamic probability update:** In the semantic thread, we only perform instance segmentation on representative frames (Key frames) and update the dynamic probability of map points according to the a priori dynamic target mask and the location relationship of projection points. The specific dynamic probability update strategy [19] is as follows:

\[
\begin{align*}
  P_{d, \text{temp}} &= \epsilon_d \cdot P_{d, \text{last}} \\
  P_{s, \text{temp}} &= \epsilon_s \cdot P_{s, \text{last}} \\
  \epsilon &= \frac{P_{d, \text{temp}}}{\epsilon_d + P_{s, \text{temp}}} \\
  P_{d, \text{current}} &= \epsilon \cdot P_{d, \text{temp}} \\
  P_{s, \text{current}} &= \epsilon \cdot P_{s, \text{temp}}
\end{align*}
\]  

(2)

where \( P_{d, \text{last}} \) and \( P_{s, \text{last}} \) are the dynamic and static probability of the last update, respectively. \( P_{d, \text{current}} \) and \( P_{s, \text{current}} \) are the dynamic and static probability, respectively, of the current key frame after the update based on instance segmentation. \( \epsilon \) is the normalization coefficient. \( p \) is the update factor. We set \( p \) to 0.7. The sum of \( \epsilon_d \) and \( \epsilon_s \) is 1. When the projection point is in the a priori dynamic region, \( \epsilon_d \) is equal to \( p \); when the projection point is outside the a priori dynamic region, \( \epsilon_s \) is equal to \( p \).
In the updating process of dynamic probability, the algorithm involves the projection of map points. The projection here refers to the projection of a point in the world coordinate system to the pixel coordinate system. The relevant formula is as follows:

\[
P_{uv} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \text{def} = \frac{1}{Z} KP = \frac{1}{Z} KTP_w
\]

where \( P_{uv} \) corresponds to the projection point on the pixel coordinate system. \( P \) corresponds to a point on the camera coordinate system. \( P_w \) corresponds to the map point on the world coordinate system. \( K \) corresponds to the internal parameter matrix of the camera. \( T \) corresponds to the transformation matrix from the world coordinate system to the camera coordinate system.

**Dynamic status update:** As shown in Figure 4, we consider a map point as dynamic when its dynamic probability is greater than 0.75. In the tracking thread, we perform a preliminary filtering operation on the projection matching results of dynamic map points.

\[
M_{\text{in-view}}^{\text{dynamic}} \text{ is the collection of all map points with matching relationships within the field of view of the current key frame to be optimized. } M_{\text{project}}^{\text{dynamic}} \text{ is the set of projection positions of } M_{\text{in-view}}^{\text{dynamic}} \text{ on the current key frame. } m_{\text{sem}}^{\text{dynamic}} \text{ and } m_{\text{dynamic}}^{\text{dynamic}} \text{ are member variables of map points, corresponding to semantic attribute probability and dynamic probability, respectively. } R_{\text{sem}}^{\text{dynamic}} \text{ is the collection of all semantic regions of the current key frame. } R_{\text{dynamic}}^{\text{dynamic}} \text{ and } R_{\text{relate}}^{\text{dynamic}} \text{ are the a priori dynamic and the a priori static regions of the current key frame, respectively. } K_{\text{sem}}^{\text{dynamic}} \text{ is the set of key frames awaiting instance segmentation.}
\]

Figure 4. Correspondence between dynamic states of map points and dynamic probabilities.

As shown in Equation (1), the time consumed by the SLAM system for pose estimation completely excludes the effect of the instance segmentation consumption time by the design of parallel processing. However, the update of the dynamic probability of map points depends entirely on the instance segmentation results of key frames, which poses the following two problems.

1. The map points detected by key frames do not cover all map points generated by normal frames. Therefore, the map point probability update is not comprehensive.
2. The processing speed of normal frames in the tracking thread is higher than that of key frames in the semantic thread. Therefore, there is a certain lag in updating the probability of map points.

Compared to the sequential execution method, the above two shortcomings make it difficult to completely filter the projection points from dynamic targets during normal frame processing. Therefore, according to the characteristics of parallel processing methods, we introduce a secondary filtering method based on regional dynamic degree and geometric constraints in Section 3.3.
3.3. Filtering Method Based on the Dynamic Degree of the Region

3.3.1. Regional Dynamic Degree Acquisition

As shown in Figure 5a, we divide the image region into m*n grid regions, inspired by the quadtree approach, to assign feature points. According to the dynamic state of the map points, we can detect the projection points for some dynamic map points. The red points in the image are the corresponding projection points. We label the grid area where the dynamic map points are projected as the dynamic area. The yellow grid area in the image corresponds to the dynamic area. As analyzed above, the dynamic status update of map points is lagging and incomplete. Therefore, when the current frame is divided into dynamic regions, sometimes a problem arises in that there are too few dynamic regions. Owing to the high frame rate of the camera, the a priori dynamic target displacement between adjacent frames is usually small. Therefore, in the short term, the historical dynamic region has a strong reference. In the SOLO-SLAM system, we save the dynamic regions of the first two frames and fuse them with the current frame. As shown in Figure 5b, the fused dynamic regions have richer information.

![Figure 5](image-url)  
*Figure 5. Regional dynamic degree acquisition. (a) corresponds to the dynamic point projection distribution of the last three frames. (b) corresponds to the projection distribution of dynamic points after merging the last three frames. (c) corresponds to the regional dynamic degree of the current frame.*

---

**Algorithm 1 Update dynamic probability and semantic properties of map points**

```
Input: Min_view Koptmize
Output: M^project R^sem
For k^optimize in K^optimize do :
    // Get instance segmentation results based on ROS
    R^sem ← RequestInstanceSegmentation(k^optimize);
    dynamic ← GetDynamicRegion(R^sem);
    static ← GetStaticRegion(R^sem);
    M^project ← GetProjectionOfMappoints (M^in_view);
    // Update only the semantic attributes of map points
    // whose projections lie in static regions
    For m^project in M^project do :
        m^dynamic ← UpdateDynamicProbability (m^project, dynamic);
        If (m^project in static) then
            m^sem ← UpdateSemanticProperty(m^project, static);
        End If
    End For
End For
```
The a priori dynamic target usually corresponds to a whole region. The grid region closer to the dynamic region is more likely to be part of the region where the a priori dynamic target is located. Therefore, we divide the dynamic degree of the grid region based on the distance from the dynamic grid region. As shown in Figure 5c, the dynamic regions are highlighted in red color. As the distance increases, it gradually changes to a cooler color. By dividing the grid area, the computational effort can be greatly reduced, and the robustness is enhanced.

After obtaining the dynamic degree of the regions based on distances, we need to convert the distances into proportions. The proportions are used for the threshold division of different regions in the subsequent operation. The conversion is performed using the sigmoid logistic regression function. The formula is

\[
\text{thr}_{\text{scale}} = \frac{1}{1 + e^{-\delta \cdot d}}
\]

where \(d\) is the distance from the dynamic grid region. \(\delta\) indicates the transformation coefficient. Thereby, we can obtain information about the scale of the distance and use it for the threshold setting.

### 3.3.2. Outlier Determination Based on the Epipolar Constraint

As shown in Figure 6, \(I_1\) and \(I_2\) are RGB images obtained by a camera at the previous and current moments. \(O_1\) and \(O_2\) are the corresponding optical center positions of the camera at the previous and current moments, respectively. \(P\) is a map point that is jointly observed under adjacent moments. Theoretically, the projection of a map point at the current moment will be on a ray consisting of poles \(e_2\) and \(p_2\) if the attitude is absolutely accurate. However, owing to the motion of dynamic objects, the projection of dynamic map points as anomalous data does not satisfy the above constraint. Moreover, the projection corresponding to the dynamic map point is usually at a farther distance from the ray \(e_2p_2\) compared to normal data. The projection point \((p_{11})\) is shown in the image. Once we obtain the dynamic degree of the region, we can further filter outliers based on the distance between the projection point and the ray.

![Figure 6. The epipolar constraint.](image)

After the initialization is completed, we perform tracking based on the motion model approach. The pose obtained from the calculation of the last frame is used as the initial value of the pose. Therefore, it is not necessary to calculate the fundamental matrix to solve for the initial value of the pose. Instead, the construction of the epipolar constraint relies on the fundamental matrix. Therefore, to perform the filtering operation in combination with estimating the distance and the dynamic degree for the regions, our work is divided into two steps.
1. Obtain the basis matrix: We first perform a preliminary matching based on Harris corner detection. Subsequently, the matching results are further optimized based on the optical flow pyramid. We also filter the points with large luminance differences in the $3 \times 3$ grid region \[16\] around the corner point locations. Finally, we calculate the most representative fundamental matrix $F$ with the RANSAC outlier filtering algorithm.

2. Obtain the distance information: the distance here refers to the distance of each projection point from the ray ($\vec{e}_2 \vec{p}_2$) at the current frame. After obtaining the fundamental matrix, we use the following equation to compute the feature points in Figure 6.

$$p_1 = [u_{p_1}, v_{p_1}]$$
$$p_\pi = [u_{p_\pi}, v_{p_\pi}]$$

where $u$ and $v$ correspond to the image pixel coordinates.

The homogeneous coordinates of $p_1$ and $p_\pi$ are expressed as

$$p_1' = [u_{p_1}, v_{p_1}, 1]$$
$$p_\pi' = [u_{p_\pi}, v_{p_\pi}, 1]$$

where $l_2$ is $\vec{e}_2 \vec{p}_2$ in Figure 6. Theoretically, the projection point should fall on $l_2$. $F$ is the fundamental matrix between adjacent frames.

$$l_2 = \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = F \cdot (p_\pi')^T = F \begin{bmatrix} u_{p_\pi} \\ v_{p_\pi} \\ 1 \end{bmatrix}$$

Based on the formula for the distance from a point to a line, we can obtain

$$\text{dis}(p_\pi, l_2) = \frac{|p_\pi' \cdot F (p_1')^T|}{\sqrt{\|X_2\|^2 + \|Y_2\|^2}}$$

By iterating through the projection points of all map points in the current frame, we can calculate the distances for all projection points. After arranging the distances from small to large, we can get the distance set $V_{\text{dis}} = \{\text{dis}_1, \text{dis}_2, \ldots, \text{dis}_n\}$. Each map point projection selects the filtering threshold in $V_{\text{dis}}$ in proportion according to $\text{thr}_{\text{scale}}$ obtained through Equation (4). When the distance corresponding to the map point is greater than the threshold and the corresponding state is static, it is preliminarily determined that the current static map point is a dynamic map point that has not been updated. The corresponding projection points are added to the set $V_{\text{outlier}}$ for filtering decisions based on depth information.

3.3.3. Outlier Determination Based on Depth Error

For tracking, we can obtain the position of the camera in the world coordinate system at each time. Simultaneously, the system continuously generates map points that have world coordinates. Theoretically, there is fixed distance information between any two known points in 3D space. However, as shown in Figure 7, at a certain historical time $t'$, the camera captures the image $I'$. Simultaneously, the system generates a map point $P_{d'}$. With the continuous movement of the camera, at the current time $t$, the camera captures the image $I$. If $P_{d'}$ is a dynamic map point, $P_{d'}$ will move in the time interval from $t'$ to $t$. The feature points on our matching are actually the projection points $p_{d\pi}$ corresponding to the map points that have moved to $P_d$. 

Simultaneously, the system generates a map point that will move in the time interval. If we use the depth image obtained at the current time to determine the corresponding pixel positions on the depth image to estimate the corresponding depth values \( \text{dep} \). According to the 3D coordinates of the map point \( P_d' \) generated in the past and the camera position of the previous frame, we can calculate the distance \( \text{dep}' \) between the two points. The formula is expressed as follows.

\[
\text{dep}' = \sqrt{(P_{cw} - P_d')^2} \\
\text{dep} = \text{dep}(P_{dn})
\]

where \( P_{cw} \) is the coordinates of the camera at the previous frame. \( P_d' \) is the coordinates of the map point itself. During the operation of the SLAM system, both \( P_{cw} \) and \( P_d' \) are known. We only need to calculate the distance between the two points. \( P_{dn} \) is the projection matching point. \( \text{dep}(x) \) refers to the search operation at the corresponding pixel coordinates in the depth image. We can get \( \text{dep} \) by finding the depth value at the pixel coordinates corresponding to \( P_{dn} \) on the depth images.

Then, we calculate the ratio of the current depth to the error.

\[
\text{dep}_{err} = \frac{\text{dep}}{|\text{dep} - \text{dep}'|}
\]

By traversing all the matching points in \( V_{\text{outlier}} \), we obtain the corresponding \( \text{dep}_{err} \) and set the empirical threshold \( \text{dep}_{\text{thr}} \). To improve the accuracy of dynamic scenes, dynamic feature points should be filtered and effective matching should be retained as much as possible to construct constraints. Therefore, when \( \text{dep}_{err} \) is greater than \( \text{dep}_{\text{thr}} \), we remove the corresponding matching point from \( V_{\text{outlier}} \). After two rounds of filtering determination, the matching points in \( V_{\text{outlier}} \) are treated as dynamic points in the pose estimation for the current frame. Thereby, we can achieve better filtering performance. Figure 8 presents a direct comparison of filtering performance. Figure 8a,b are pictures at are two adjacent frames. We can see that (b) presents an extreme case. Owing to the lag in the dynamic information update of map points, there is no reference dynamic map point projection in (b). Through the division of regional dynamic degrees based on historical information, we further filter the projection points of some a priori dynamic regions through two rounds of employing constraints.
Figure 8. Further comparison of filtering performance. In the figure, (a,b) are two adjacent frames. (c,d) are randomly extracted frames. Red points represent the projections of dynamic map points. Blue points represent the projections of static map points. Green points are the projections of filtered map points in the second round. The points selected in the yellow rectangular box correspond to the green points in the a priori dynamic region.

3.4. Constraint Construction Based on Semantic Information

We have added semantic threads and semantic optimization threads to SOLO-SLAM. The semantic thread is responsible for updating the dynamic probability and semantic attributes of the map points based on instance segmentation. After obtaining the semantic attributes of the map points and the semantic masks of the key frames, we combine them in the semantic optimization thread to build a new optimization method. Next, we describe the specific implementation details.

3.4.1. Implementation Details of the Semantic Threads

Instance segmentation method: In the SOLO-SLAM system, we perform mask acquisition using a DL algorithm after creating key frames. As shown in Figure 9, to obtain the semantic information of the image, we use the state-of-the-art instance segmentation algorithm (SOLO_V2) to perform the image segmentation task. The core idea of SOLO_V2 is to transform the segmentation problem into a positional classification problem. It innovatively decouples the original mask into a mask kernel branch and a mask feature branch, which are used to predict the convolutional kernel and convolutional feature, respectively. After improvement, SOLO_V2 shows excellent performance. Details about SOLO_V2’s usage are as follows:

- SOLO-SLAM is designed based on the ROS system. We designed the ROS interface program for SOLO_V2. The data interaction with the SLAM part is realized by encapsulating SOLO_V2 as a function package.
• The SOLO_V2 algorithm is trained on the COCO dataset [25] and can recognize up to 80 categories. The recognized objects include people, bicycles, cars, and other potential motion objects. The plentiful recognizable categories can help us to recognize both common a priori dynamic objects and environmental information such as TVs, chairs, and bottles in the scene. This can provide sufficient support for our semantic constraint construction.

• The SOLO-SLAM system runs on the PyTorch DL framework. The application of SOLO_V2 algorithm is implemented with the help of MMdetection Python toolbox.

• The instance segmentation algorithm in SOLO-SLAM uses a 50-layer backbone network (solov2_light_512_dcn_r50_fpn_8gpu_3x.py) provided by the authors of SOLO-V2. The training and use involved in the SOLO-SLAM system are implemented based on this network structure.

![Figure 9. Processing operations in semantic threads.](image)

**Mask usage details:** After obtaining the instance results via SOLO-V2, SOLO-SLAM first updates the probability of map points based on the projected points in the a priori dynamic region. After the dynamic probability update is completed, the mask of the a priori static region is traversed, and the semantic attributes of the map points are updated according to the mask class of the region where the projection points are located.

\[
\begin{align*}
    p_{sem1\_current} &= \left[p_{sem1\_last} + (1 - \epsilon_{sem}) \right] p_{flag} \\
    p_{sem2\_current} &= \left[p_{sem2\_last} + (1 - \epsilon_{sem}) \right] p_{flag} \\
    \vdots \\
    p_{semn\_current} &= \left[p_{semn\_last} + (1 - \epsilon_{sem}) \right] p_{flag}
\end{align*}
\]

where \( p_{semn\_current} \) is the probability corresponding to the nth semantic attribute of the current map point. We take the semantic attribute with the largest probability value as the semantic attribute of the current map point. \( \epsilon_{sem} \) is the update factor, which can be set to any value less than 0.5 based on experience. When the map point projection falls in the corresponding semantic region, \( p_{flag} \) is set to 1. Otherwise, \( p_{flag} \) is set to 0. In contrast to the dynamic probability update method, when the projection corresponding to the map point is in the nth new semantic region for the first time, we set \( p_{semn\_current} \) equal to 0.5. Afterwards, the update is performed according to Equation (9). We maintain the update of only two semantic attributes in SOLO_SLAM. When it falls in the third semantic region, we consider that the semantic attribute of the map point has extremely low confidence. Consequently, the corresponding map point will no longer participate in semantic optimization.

**3.4.2. Implementation Details of the Semantic Optimization Thread**

**Semantic constraint principle:** Objects in real-world environments have semantic information. The ORB corner points [26] detected from the objects by the SLAM system also have semantic properties corresponding to the objects. As shown in Figure 10a, the instance segmentation algorithm allows us to assign semantic labels to objects in the
environment. Combining the semantic labels of objects and the semantic attributes of map points, we can impose constraints. Taking the monitor as an example, theoretically, when the pose is absolutely accurate, the projections of map points from the monitor at the current frame should necessarily be inside the semantic region corresponding to the monitor. However, owing to pose errors, the projections of map points are often scattered near the semantic region as shown in Figure 10c. Therefore, it can be used as a constraint that the projections of map points with semantic attributes should be inside the region with the same semantic labels.

Figure 10. Semantic constraint principle. (a) shows the result of the Instance segmentation. (b) shows the result after being converted into a grayscale image. (c) is a schematic diagram of projection points.

Construction of semantic constraints: SLAM systems based on feature point methods usually use the minimization of the re-projection error to impose constraints. In the tracking process, we can obtain the projections of map points in the current frame and the corresponding matching points. Consequently, the pose and map points can be continuously optimized by methods such as graph optimization or extended Kalman filtering. The paper uses a graph optimization method, i.e., a nonlinear optimization method. The code implementation is based on the g2o [27] functional package. In this paper, the graph optimization method is designed to use the 3D coordinates of map points obtained during the operation of the system and the pose at each moment as optimization vertices respectively. At the same time, constraints are constructed as edges connecting vertices. Our main work lies on the design of new constraints. The quantitative index in the optimization process is the distance of the projection point from the matching point. By minimizing the distance, the SLAM system can improve the system accuracy. Similarly, we transform the semantic constraint into the error between 2D coordinates and minimize the distance as a quantitative indicator. Two constraints are represented as follows:

\[
\begin{align*}
\text{err}_{\text{re proj}} &= K \cdot T_{cw} \cdot P - p_{\text{match}} \\
\text{err}_{\text{sem proj}} &= K \cdot T_{cw} \cdot P - p_{\text{nearest}}
\end{align*}
\]

where \(\text{err}_{\text{re proj}}\) is the constraint constructed based on the re-projection error. \(\text{err}_{\text{sem proj}}\) is the constraint constructed based on semantic information. \(T_{cw}\) is the transformation matrix. \(P\) represents the map points. \(p_{\text{match}}\) represents the matching points. As we can see, the key is the acquisition of \(p_{\text{nearest}}\).

In the semantic thread, we save the static region mask corresponding to each key frame. The acquisition of the mask is dependent on the mask information. The relevant process can be divided into four steps as follows.

Step 1: Iterate through the mask categories of all the a priori static regions and calculate the center of mass for the mask regions. Since instance segmentation can distinguish between different individuals of the same category, the search area can be further refined.

Step 2: Determine whether there is more than one mask region in the same category. When there are multiple mask regions, the mask with the closest distance to the projection point is selected as the target mask.

Step 3: Evaluate the position relationship with the semantic region according to the projected position.
Step 4: Extract the target region contour. Search for the nearest neighboring point on the contour as $p_{\text{nearest}}$.

In SOLO-SLAM, we perform graph optimization based on g2o. While delineating edges, we update the position relations in step 3 in real time. When an object is inside the semantic region, we temporarily remove the corresponding edge in this optimization round. The overall optimization is performed simultaneously with the re-projection error-based optimization. The cost function at the final optimization is defined as

$$T = \arg\min \sum_{i} \text{err}_{\text{reproj}}(k, i) + \lambda \sum_{i} \text{err}_{\text{semproj}}(k, i)$$ (13)

where $T$ is the pose to be optimized, $k$ is the index of the key frame, $i$ is the index of all map points in the current key frame, and $\lambda$ is the weighting factor. We set $\lambda$ to 0.3. Using the above method, we introduce more constraints to facilitate optimization. As shown in Algorithm 2, we also provide pseudocode to demonstrate the process. In the next section, we compare our system with other state-of-the-art algorithms to demonstrate its overall performance.

Algorithm 2 Update Nearest Point of Map Point

| Input   | $\text{M}_{\text{in view}}$, $\text{M}_{\text{project}}$, $\text{R}_{\text{sem}}$ |
|---------|-------------------------------------------------|
| Output  | $\text{M}_{\text{in view}}$ |

For $k_{\text{optimize}}$ in $K_{\text{optimize}}$ do:

For $m_{\text{in view}}$ in $\text{M}_{\text{in view}}$ do:

$m_{\text{project}} \leftarrow \text{GetProjectionPoints}(\text{M}_{\text{project}})$;

$m_{\text{sem}} \leftarrow \text{GetSemanticProperties}(m_{\text{in view}})$;

// Iterate over the semantic region of the current frame

// to get the nearest point

For $r_{\text{sem}}$ in $\text{R}_{\text{sem}}$ do:

$\text{sem}_\text{temp} \leftarrow \text{GetRegionSemanticProperties}(r_{\text{sem}})$;

If $(m_{\text{sem}} == \text{sem}_\text{temp})$ then

$p_{\text{temp}} \leftarrow \text{GetNearestPointOfContours}(m_{\text{project}})$;

$p_{\text{nearest}} \leftarrow \text{UpdateNearestPoint}(m_{\text{in view}} \rightarrow p_{\text{nearest}}, p_{\text{temp}})$;

End If

End For

End For

SemanticOptimizationViaG2O($\text{M}_{\text{in view}}$, $\text{M}_{\text{project}}$);

End For

where $\text{M}_{\text{in view}}$ is the set of all map points with matching relationships within the field of view of the current key frame to be optimized. $\text{M}_{\text{project}}$ is the set of projected positions of $\text{M}_{\text{in view}}$ on the current key frame. $\text{R}_{\text{sem}}$ is the set of all semantic regions of the current key frame. $m_{\text{sem}}$ and $p_{\text{nearest}}$ are member variables of map points. $K_{\text{optimize}}$ is the set of key frames awaiting semantic optimization.

4. Experimental Results and Discussion

In this section, we demonstrate the effectiveness of our algorithm by conducting experimental tests on a publicly available dataset. We compare our SLAM system with other state-of-the-art SLAM systems in terms of accuracy and real-time performance.

4.1. Dataset and Algorithms for Comparison

Our proposed SOLO-SLAM system is an RGBD image-based SLAM system for dynamic scenes. Therefore, we select the public TUM dynamic objects dataset [28] for evaluation. The associated dataset was produced based on a kinect camera. Since the kinect camera is based on TOF technology for depth acquisition, we can get a series of $640 \times 480$ color and depth images. The corresponding datasets are divided into two categories based on the dynamic degree of scenes: $w$, which stands for walking, corresponding to a highly dynamic scene; and $s$, which stands for sitting, corresponding to a low dynamic scene. The specific scene is two people sitting at a table, chatting and gesturing. Based on the
type of camera motion, the dataset is divided into four categories: (1) static: the sensor is kept in place manually; (2) xyz: the sensor moves along three directions (xyz) while keeping the same orientation; (3) halfspare: the sensor moves on a small half sphere of approximately one meter diameter; and (4) rpy: the sensor rotates along the principal axes (roll–pitch–yaw) at the same position. We provide the comparative results of SOLO-SLAM with other algorithms on the above dataset below.

In the selection of algorithms for comparison, on the one hand, ORB-SLAM3 is considered the most advanced algorithm for static scenes at present; on the other hand, our SOLO-SLAM system is based on ORB-SLAM3 and aims at improving its performance. Therefore, we first selected ORB-SLAM3 as one of the comparison algorithms. In addition, we also selected four SLAM algorithms for comparison with respect to dynamic scenes. Among them, Dyna-SLAM and DS-SLAM are two well-known advanced open-source SLAM systems. In Section 2, we discussed Detect-SLAM and RDS-SLAM from the perspective of applying the dynamic probability of map points, so we also included them in the comparison to demonstrate the effectiveness of our proposed algorithm.

4.2. Tracking Accuracy Evaluation

The absolute trajectory error (ATE) corresponds to the difference between the true value of the camera pose and the estimated value of the SLAM system. This criterion is well suited to evaluating the performance of visual SLAM systems. Tables 1 and 2 show the ATE results of the algorithm under different scenarios. We used the accuracy data provided in the relevant studies as comparison data. For the data not provided in the Dyna-SLAM and DS-SLAM studies, we ran the open-source code and selected the mean values after multiple tests for comparison. For unavailable data, we used "-" to indicate them.

| Sequences | ATE [m] |
|-----------|---------|
| fr3_w_xyz | 0.6580  |
| fr3_w_static | 0.2648  |
| fr3_w_rpy | 0.5404  |
| fr3_w_hs  | 0.2957  |
| fr3_s_hs  | 0.0193  |

Table 1. Experimental results of absolute trajectory error (ATE [m]).

The root mean square error (RMSE) and standard deviation (SD) values are reported in the tables to better demonstrate the robustness and stability of the system. The mean is the mean value of errors in the table, which can better reflect the accuracy information of the system. Referring to the information in the tables, we can see that the accuracy of SOLO-SLAM is improved by one order of magnitude compared to ORB-SLAM3 on the highly dynamic dataset. Among them, the mean and RMSE show the highest improvements of 97.46% and 97.63%, respectively, in highly dynamic scenarios, and 37.84% and 43.55%, respectively, in low dynamic scenarios. This fully validates the effectiveness of SOLO-SLAM in improving system accuracy and stability in highly dynamic scenes. Additionally,
this proves the rationality of the idea of parallel processing based on the dynamic state of map points. Compared to the results of the original studies of Detect-SLAM, RDS-SLAM, and DS-SLAM, our algorithm achieves a significant accuracy improvement on almost all datasets tested.

This is most likely due to two reasons. First, the secondary filtering method based on the dynamic degree of the region mitigates the problem of untimely and insufficient dynamic state updates of map points. Second, the SOLO-V2-based instance segmentation mask can obtain more accurate semantic information compared to the target detection frame obtained from SSD. Unfortunately, although the accuracy of the SOLO-SLAM system has been greatly improved, there is still a gap between it and Dyna-SLAM under highly dynamic scenarios. This is because Dyna-SLAM is a sequential processing structure that can fully acquire the a priori dynamic regions in each frame. Dyna-SLAM achieves the best dynamic point filtering performance by combining segmentation results and the region growth algorithm. In contrast, the dynamic point filtering capability of SOLO-SLAM’s parallel processing approach is limited. However, it is worth noting that the SOLO-SLAM system has the highest accuracy in low dynamic scenes compared to all algorithms tested in this study. This is because the ORB-SLAM3 mechanism based on RANSAC and co-view can filter some of the anomalous dynamic points in low dynamic scenes to a certain extent. Nevertheless, the direct filtering of the a priori dynamic region feature points will result in losing many feature points in low dynamic scenes, leading to fewer constraints that can be constructed during local optimization. In contrast, the SOLO-SLAM system adds constraints to the optimization process based on the semantic attributes of map points, which improves the accuracy in low dynamic scenes to a certain extent.

The relative pose error (RPE) is the difference in the amount of positional change within the same two time stamps. This criterion is suitable for estimating the drift of the system. Tables 3 and 4 show the comparison results in terms of both translations and rotations, respectively. The increments between the relative positions used to calculate the errors in the table are set to 0.1 m. The results are similar to the ATE results. It is worth noting that sequences based on pure rotational motion (rpy) have a relatively low lifting effect. This is probably due to the presence of a large number of 45° rotations of the sensor, which lead to a more difficult update of the dynamic state and semantic attributes of the map points. Compared to methods such as Dyna-SLAM with sequential execution, some dynamic feature points are not effectively eliminated. We also provide estimation results of different algorithms and the ground truth in Figure 11 to provide intuitive comparison.

Table 3. Experimental results of translational relative trajectory error.

| Sequences | ORB-SLAM3 | Improvements against ORB-SLAM3 | Dyna-SLAM | DS-SLAM | Improvements against DS-SLAM | SOLO-SLAM |
|-----------|------------|-------------------------------|-----------|---------|-------------------------------|-----------|
| fr3_w_xyz | 0.2033     | 0.0225                        | 91.54%    | 91.11%  | 0.0144                        | 0.0165    |
| fr3_w_stat| 0.0264     | 0.02726                       | 60.98%    | 57.08%  | 0.0078                        | 0.007     |
| fr3_w_rpy | 0.0794     | 0.0994                        | 60.83%    | 54.53%  | 0.0279                        | 0.0363    |
| fr3_w_hs  | 0.0379     | 0.0512                        | 52.77%    | 58.20%  | 0.0166                        | 0.0192    |
| fr3_s_hs  | 0.0139     | 0.0169                        | 72.72%    | 4.14%   | 0.0197                        | 0.0025    |

Table 4. Experimental results of rotational relative trajectory error.

| Sequences | ORB-SLAM3 | Improvements against ORB-SLAM3 | Dyna-SLAM | DS-SLAM | Improvements against DS-SLAM | SOLO-SLAM |
|-----------|------------|-------------------------------|-----------|---------|-------------------------------|-----------|
| fr3_w_xyz | 0.1461     | 0.1777                        | 91.99%    | 89.87%  | 0.0097                        | 0.0109    |
| fr3_w_stat| 0.1344     | 0.159                         | 94.64%    | 95.03%  | 0.0095                        | 0.006     |
| fr3_w_rpy | 0.1221     | 0.1545                        | 63.47%    | 10.53%  | 0.0164                        | 0.0219    |
| fr3_w_hs  | 0.0232     | 0.0334                        | 44.40%    | 55.09%  | 0.0128                        | 0.0149    |
| fr3_s_hs  | 0.012717   | 0.014356                      | 1.71%     | 2.48%   | 0.0175                        | 0.0253    |
Figure 11. Track diagrams of different dynamic scenes. ORB-SLAM3 corresponds to (a–e); Dyna-SLAM corresponds to (f–j); DS-SLAM corresponds to (k–o); SOLO-SLAM corresponds to (p–t). The red line corresponds to the difference between the ground truth and the calculated trajectory.

4.3. Real-Time Evaluation

In terms of equipment, both SOLO-SLAM and Dyna-SLAM were tested using a laptop computer. The CPU of the computer was an Intel-i7-12700H with an NVIDIA-3070TI graphics card. Owing to the cuDNN version required for DS-SLAM configuration, we ran the algorithm with an NVIDIA-1060 graphics card. Therefore, we only compared the real-time performance of Dyna-SLAM and SOLO-SLAM on the same device.

SOLO-SLAM adopts a parallel processing approach. Theoretically, the time consumed by SOLO-SLAM is completely independent of the time consumed by instance segmentation. However, the fast processing of images by the tracking threads may result in significant lag in updating the dynamic state and semantic attributes of map points, which will further affect the accuracy of the system. Therefore, the results of our tests were controlled for using a processing speed of 15 frames per second. The implementation was conducted by adding a time delay between frames.

We provide the difference between the current frame ID being processed in the test process and the key frame ID that had just completed the instance segmentation. We hope that Figures 12 and 13 can give other researchers an intuitive understanding of hysteresis. When performing instance segmentation on key frames that have not yet been instance
segmented, we take turns to select the most recently obtained key frame and the earliest key frame for segmentation. Therefore, we can see that there are two corresponding graphs. Figure 12 corresponds to the minimum difference in processing each frame with an average lag of 6.8 frames Figure 13 corresponds to the maximum difference in processing each frame with an average lag of 10.2 frames.

Figure 12. The difference between the nearest keyframe ID and the current frame ID.

Figure 13. The difference between the oldest keyframe ID and the current frame ID.

Table 5 shows the real-time improvement effect of SOLO-SLAM’s parallel processing approach compared to Dyna-SLAM. Combining 3.2 and 3.3, we can see that our algorithm achieves better accuracy improvement compared to sequential processing methods, such as DS-SLAM. Further, compared to Dyna-SLAM, SOLO-SLAM achieves better accuracy in low dynamic scenes. Although there is still a slight difference in accuracy in high dynamic scenes, SOLO-SLAM is still of practical importance due to the substantial improvement in real-time performance.

Table 5. Comparison of Dyna-SLAM and SOLO-SLAM processing times.

| Sequences     | fr3_w_xyz | fr3_w_static | fr3_w_rpy | fr3_w_hs | fr3_s_hs |
|---------------|-----------|--------------|-----------|----------|---------|
| Dyna-SLAM [ms]| 674.68    | 587.60       | 574.66    | 643.86   | 578.68  |
| SOLO-SLAM [ms]| 67.00     | 67.00        | 67.00     | 67.00    | 67.00   |
| Improvements against DynaSLAM | 90.07% | 88.60% | 88.34% | 89.59% | 88.42% |
4.4. Evaluation in Real Environment

In order to further prove the effectiveness of the algorithm, we carried out experimental verification using a realsense d435 camera in real scenes. The resolution of the camera is 640 * 480. The acquisition frame rate of the camera is set to 30 frames per second. Relying on the realsense d435 camera, we can obtain the RGB image and depth image of the scene.

Our verification is divided into two parts.

First, we tested the filtering performance of SOLO-SLAM against dynamic points in different scenarios. The scene is divided into indoor and outdoor according to the location, and the scene is divided into normal light intensity environment and weak light intensity environment according to light intensity. Figure 14 shows the intuitive filtering effect. We can see that almost all the characteristic points on pedestrians are marked as red or green. The red dots in the figure mean the dots filtered out in the first round of filtering. The green dots in the figure mean the dots filtered out in the second round of filtering. SOLO-SLAM filters out almost all feature points in the prior dynamic region.

We also tested ORB-SLAM3 and SOLO-SLAM in the indoor environment based on the realsense d435 camera. In the test scenario, there are walking pedestrians in front of the camera. Because we could not obtain the true value of the trajectory in the real scene, we
tried to make a loop motion indoors. To put this simply, we used a starting point and an ending point located at the same position as much as possible. As can be seen in Figure 15, the trajectory of SOLO-SLAM almost constitutes a loopback. In contrast, ORB-SLAM3 undergoes a significant offset.

![Comparison of ORB-SLAM3 and our system in indoor environment.](image)

**Figure 15.** Comparison of ORB-SLAM3 and our system in indoor environment.

5. Conclusions

In this study, we introduced a novel dynamic point filtering method based on ORB-SLAM3 with parallel processing to solve the problem of the poor real-time performance of traditional schemes as represented by Dyna-SLAM. Using the SOLO-V2 function package designed based on the ROS system, we provided semantic information for the SLAM system by combining the instance segmentation results of key frames. For semantic information, we put forward three improvements and tested them against other state-of-the-art SLAM algorithms on the TUM dynamic object dataset. The experimental validation allows us to draw the following conclusions:

1. We implemented the update of dynamic probability and semantic attributes of map points based on the instance segmentation method in a new semantic thread. Simultaneously, adopting a motion model-based tracking approach, we performed dynamic point filtering based on the dynamic state of the projected points. This approach avoids the shortcomings of the traditional sequential processing approach by reducing the processing time for obtaining the a priori dynamic regions. By comparison, our system achieves a significant improvement in real-time performance.

2. We proposed a quadratic filtering method based on the regional dynamic degree and geometric constraints to address the problem of inadequate dynamic point recognition under the parallel processing approach. By comparison, the accuracy of our system outperforms other algorithms, including the DS-SLAM scheme with sequential processing in most cases.

3. We combined the semantic attributes of map points and the semantic information of the current frame to increase the constraint term during optimization. The proposed algorithm solves the problem of accuracy degradation caused by excessive dynamic point filtering in low dynamic scenes to a certain extent. By comparison, SOLO-SLAM achieves the best accuracy in low dynamic scenes.

Although SOLO-SLAM already has good performance in dynamic scenarios, we believe that there are still possibilities for further improvements.

We note that in the low dynamic scenes in the TUM dataset, most parts of the human body are static. Based on common sense, most of the vehicles in the parking lot remain static. In the above example, directly and brutally filtering the feature points in the a priori dynamic region will lead us to lose a great deal of information that can be used. Therefore,
we believe that effectively distinguishing the static parts in the a priori static region is a practical idea for future improvement.

In addition, we think it will be interesting to combine this with other sensors. In the follow-up work, we suggest improving it in combination with wheel odometry and lidar. At present, the visual SLAM algorithm is limited by the strong assumption of a static environment. When there are moving objects in the environment, the accuracy of the system will be seriously reduced due to the large number of abnormal values. In contrast, the wheel odometry obtains the motion state based on the track estimation of the wheel. This method has high locating accuracy in a short time and short distance and is not affected by dynamic targets in principle. However, the wheel odometry has some problems; for example, the accumulated error cannot be eliminated and the wheel slip introduces abnormal information. Therefore, the tight coupling of the vision method and wheel odometry method to achieve complementarity has objective and practical significance for improving system performance.

In the subsequent work, we also plan to carry out an improved solution based on the combination of vision and LiDAR. The vision SLAM and laser SLAM are similar in principle in the back-end optimization part. However, in the front-end part, LiDAR can directly provide point cloud information in the environment. These point cloud features are almost unaffected by light variations and are highly accurate. In addition to this, the point cloud information is returned directly from the laser scanner without additional computation. This also helps to achieve real-time scene building. Therefore, the combination of LiDAR helps us to further enhance the real-time mapping capability of the SLAM system and to overcome the effects of drastic illumination changes.

Author Contributions: Conceptualization, L.S.; methodology, L.S. and J.W.; software, L.S. and J.W.; validation, L.S., J.W. and S.S.; formal analysis, L.S.; investigation, L.S.; resources, L.S. and S.S.; data curation, L.S. and P.W.; writing—original draft preparation, L.S.; writing—review and editing, L.S. and P.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research is financially supported by the National Natural Science Youth Fund under Grant No. 62201598.

Data Availability Statement: The dataset used in the paper is the public TUM dataset. It can be downloaded at this link: https://vision.in.tum.de/data/datasets/rgbd-dataset.

Conflicts of Interest: The authors declare no conflict of interest.

References
1. Cadena, C.; Carlone, L.; Carrillo, H.; Latif, Y.; Scaramuzza, D.; Neira, J.; Reid, I.; Leonard, J.J. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. IEEE Trans. Robot. 2016, 32, 1309–1332. [CrossRef]
2. Campos, C.; Elvira, R.; Rodriguez, J.J.G.; Montiel, J.M.; Tardós, J.D. Orb-Slam3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap Slam. IEEE Trans. Robot. 2021, 37, 1874–1890. [CrossRef]
3. Qin, T.; Pan, J.; Cao, S.; Shen, S. A General Optimization-Based Framework for Local Odometry Estimation with Multiple Sensors. arXiv 2019, arXiv:190103638.
4. Engel, J.; Koltun, V.; Cremers, D. Direct Sparse Odometry. IEEE Trans. Pattern Anal. Mach. Intell. 2017, 40, 611–625. [CrossRef][PubMed]
5. Wan Aasim, W.F.A.; Okasha, M.; Faris, W.F. Real-Time Artificial Intelligence Based Visual Simultaneous Localization and Mapping in Dynamic Environments—a Review. J. Intell. Robot. Syst. 2022, 105, 1–23. [CrossRef]
6. Song, S.; Lim, H.; Lee, A.J.; Myung, H. DynaVINS: A Visual-Inertial SLAM for Dynamic Environments. IEEE Robot. Autom. Lett. 2022. [CrossRef]
7. Sun, Y.; Liu, M.; Meng, M.Q.-H. Motion Removal for Reliable RGB-D SLAM in Dynamic Environments. Robot. Auton. Syst. 2018, 108, 115–128. [CrossRef]
8. Bescos, B.; Fácil, J.M.; Civera, J.; Neira, J. DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes. IEEE Robot. Autom. Lett. 2018, 3, 4076–4083. [CrossRef]
9. Kim, D.-H.; Kim, J.-H. Effective Background Model-Based RGB-D Dense Visual Odometry in a Dynamic Environment. IEEE Trans. Robot. 2016, 32, 1565–1573. [CrossRef]
10. Sun, Y.; Liu, M.; Meng, M.Q.-H. Improving RGB-D SLAM in Dynamic Environments: A Motion Removal Approach. Robot. Auton. Syst. 2017, 89, 110–122. [CrossRef]
11. Li, S.; Lee, D. RGB-D SLAM in Dynamic Environments Using Static Point Weighting. *IEEE Robot. Autom. Lett.* **2017**, *2*, 2263–2270. [CrossRef]

12. Mur-Artal, R.; Tardós, J.D. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Trans. Robot.* **2017**, *33*, 1255–1262. [CrossRef]

13. Raguram, R.; Chum, O.; Pollefeys, M.; Matas, J.; Frahm, J.-M. USAC: A Universal Framework for Random Sample Consensus. *IEEE Trans. Pattern Anal. Mach. Intell.* **2012**, *35*, 2022–2038. [CrossRef][PubMed]

14. Pire, T.; Fischer, T.; Castro, G.; De Cristóforis, P.; Civera, J.; Berlles, J.J. S-PTAM: Stereo Parallel Tracking and Mapping. *Robot. Auton. Syst.* **2017**, *93*, 27–42. [CrossRef]

15. He, K.; Gkioxari, G.; Dollár, P.; Girshick, R. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, Venice, Italy, 22–29 October 2017; pp. 2961–2969.

16. Yu, C.; Liu, Z.; Liu, X.-J.; Xie, F.; Yang, Y.; Wei, Q.; Fei, Q. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments. In *Proceedings of the 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 1–5 October 2018; pp. 1168–1174.

17. Long, X.; Zhang, W.; Zhao, B. PSIPNet-SLAM: A Semantic SLAM Detect Dynamic Object by Pyramid Scene Parsing Network. *IEEE Access* **2020**, *8*, 214685–214695. [CrossRef]

18. Zhong, F.; Wang, S.; Zhang, Z.; Wang, Y. Detect-SLAM: Making Object Detection and SLAM Mutually Beneficial. In *Proceedings of the 2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Lake Tahoe, NV, USA, 12–15 March 2018; pp. 1001–1010.

19. Liu, Y.; Miura, J. RDS-SLAM: Real-Time Dynamic SLAM Using Semantic Segmentation Methods. *IEEE Access* **2021**, *9*, 23772–23785. [CrossRef]

20. Sucar, E.; Hayet, J.-B. Bayesian Scale Estimation for Monocular Slam Based on Generic Object Detection for Correcting Scale Drift. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 21–25 May 2018; pp. 5152–5158.

21. Zhang, T.; Zhang, H.; Li, Y.; Nakamura, Y.; Zhang, L. Flowfusion: Dynamic Dense Rgb-d Slam Based on Optical Flow. In *Proceedings of the 2020 IEEE International Conference on Robotics and Automation (ICRA)*, Philadelphia, PA, USA, 23–27 May 2020; pp. 7322–7328.

22. Sun, D.; Yang, X.; Liu, M.-Y.; Kautz, J. Pwc-Net: Cnns for Optical Flow Using Pyramid, Warping, and Cost Volume. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 18–23 May 2018; pp. 8934–8943.

23. Lianos, K.-N.; Schönberger, J.L.; Pollefeys, M.; Sattler, T. Vso: Visual Semantic Odometry. In *Proceedings of the 15th European Conference on Computer Vision (ECCV 2018)*, Munich, Germany, 8–14 September 2018; pp. 234–250.

24. Wang, X.; Zhang, R.; Kong, T.; Li, L.; Shen, C. Solov2: Dynamic and Fast Instance Segmentation. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17721–17732.

25. Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft Coco: Common Objects in Context. In *Proceedings of the European Conference on Computer Vision*, Zurich, Switzerland, 6–12 September 2014; pp. 740–755.

26. Mur-Artal, R.; Montiel, J.M.M.; Tardos, J.D. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Trans. Robot.* **2015**, *31*, 1147–1163. [CrossRef]

27. Kümerle, R.; Grisetti, G.; Strasdat, H.; Konolige, K.; Burgard, W. G2o: A General Framework for Graph Optimization. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, 9–13 May 2011; pp. 3607–3613.

28. Sturm, J.; Engelhard, N.; Endres, F.; Burgard, W.; Cremers, D. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, 7–12 October 2012; pp. 573–580.