Computers and Artificial Intelligence；Vol.3, No.2; 2026
E-ISSN: 3006 -3728 P-ISSN:3006-371X
DOI: https://doi.org/10.70267/cai.26v3n2.0112
Published by: Zeus Press

Visual SLAM Algorithm Optimized by Dynamic Feature Elimination and Keyframe Selection

Xue Zhao, Na Gao*, Qingqing Zhang, Yao Zhao, Lili Yuan and Jianye Wang

Henan Polytechnic University, Jiaozuo, Henan, China

**Corresponding author: Na Gao*

Abstract

Traditional visual SLAM algorithms struggle to maintain stability in dynamic scenes, often resulting in tracking failures and low positioning accuracy. This paper proposes the YDK-SLAM algorithm, which is suitable for dynamic environments. Building upon ORB-SLAM3, a dynamic feature detection thread has been added. It employs the YOLOv8 object detection network to identify potential moving objects, precisely removing dynamic feature points through depth information and multi-view geometry methods, and utilizing the remaining static feature points for pose estimation. Simultaneously, by employing optical flow methods and adaptive keyframe insertion strategies in the tracking thread based on camera motion characteristics, thereby enhancing the operational speed of YDK-SLAM. Experimental results on the public TUM dataset demonstrate that compared to ORB-SLAM3, YDK-SLAM achieves an average reduction of 70.73% in absolute trajectory error. In high-dynamic scenes, the error is reduced by an average of 93.73%, effectively enhancing the system's positioning accuracy and robustness.

Keywords

visual slam, dynamic feature point removal, keyframe optimization, dynamic environment

1. Introduction

Environmental perception and positioning navigation form the cornerstone for developing mobile robots and unmanned aerial vehicles. A direct and effective solution to address perception and positioning challenges in uncharted environments is Simultaneous Localization and Mapping (SLAM) [1]. Traditional visual SLAM algorithms can all achieve satisfactory performance under static conditions. However, dynamic objects present in the real world disrupt the static assumptions of traditional SLAM algorithms, leading to issues such as feature matching errors and failed geometric constraints. This results in degraded positioning accuracy or even positioning failure, significantly limiting the application of SLAM technology in real-world environments. With the advancement of deep learning technology, some researchers have applied deep learning techniques to SLAM algorithms in dynamic environments, achieving promising results by leveraging semantic segmentation to address interference from moving objects. Semantic segmentation technology performs pixel-level semantic segmentation on input images. While it enhances the accuracy of dynamic point recognition, it involves substantial computational demands and is time-consuming. Another group of researchers has proposed SLAM algorithms that utilize object detection to identify dynamic points.

These approaches not only reduce computational overhead but also offer superior real-time performance compared to semantic segmentation, making them widely adopted in dynamic SLAM applications.

Although object detection methods have low computational complexity, the detected bounding boxes are typically large. Blindly removing all feature points within a bounding box will reduce the accuracy of dynamic feature point recognition. ORB-SLAM3 [2] is the most comprehensive and robust algorithm in the ORB-SLAM series. However, it requires frame-by-frame extraction of ORB feature points and selection of keyframes. Computing the FAST corners and BRIEF descriptors for ORB feature points frame by frame introduces significant redundant computations, resulting in low overall algorithmic efficiency. To address the interference of dynamic targets on SLAM algorithms and enhance overall algorithmic efficiency, this paper proposes the YDK-SLAM algorithm. The specific research focuses include the following aspects:

(1) To prevent interference from moving objects on the SLAM algorithm, a dynamic feature detection module has been added to the ORB-SLAM3 system framework, it employs the YOLOv8 object detection network **错误!未找到引用源。** to detect, recognize, and classify image frames, when integrated with an object detection network, this algorithm can effectively detect and eliminate interference from moving objects.

(2) When using the YOLOv8 object detection network to detect moving objects, the detection typically includes portions of the background, leading to misclassification of feature point attributes. This algorithm utilizes depth information to separate the background from moving objects and employs geometric motion consistency methods to remove dynamic feature points while retaining static ones for inter-frame matching. This method not only reduces the interference of dynamic targets on camera pose estimation but also enhances the positioning accuracy of the SLAM algorithm.

(3) The computation of FAST corners and BRIEF descriptors for each frame's feature points is time-consuming. This algorithm proposes a strategy for adaptively selecting tracking methods and inserting keyframes based on the camera's motion state. When the camera moves smoothly, it employs optical flow methods to enhance computational efficiency. During rapid camera motion, it utilizes feature point method to effectively prevent tracking failures caused by the system losing scene information in complex environments.

2. Research Works

2.1 Dynamic Slam Method Combined with Deep Learning

In recent years, to address the challenges faced by SLAM algorithms in dynamic environments, many researchers have employed methods such as image semantic segmentation or object detection to remove moving objects. DS-SLAM [3] employs SegNet for semantic segmentation and utilizes motion consistency checks to eliminate dynamic elements within scenes, thereby generating comprehensive semantic tree graphs. Bescos et al. [4] employed a method combining the Mask R-CNN network with multi-view geometry to segment and remove dynamic objects. Pan et al. [5] proposed a robust dynamic feature segmentation SLAM algorithm that utilizes an optimized epipolar geometry model and combines it with Mask R-CNN for image segmentation to eliminate dynamic outliers. However, dynamic SLAM based on semantic segmentation methods imposes high computational demands on the platform. SG-SLAM [6] utilizes semantic information from the NCNN network alongside geometric information to eliminate dynamic points. The optimized model is integrated with the ORB-SLAM2 framework to enhance algorithmic speed. Su et al. [7] utilize semantic threads to extract semantic information from scenes, optimizing the homography matrix module and optical flow module to eliminate dynamic feature points. Chang et al. [8] employed the lightweight detection network YOLOv4-tiny to detect moving objects, eliminating the influence of dynamic feature points during the tracking phase. CD-SLAM [9] utilizes semantic information from the YOLOv5 network and prior information from the inertial measurement unit (IMU) to eliminate dynamic targets, while employing scene flow to compensate for potential static information. Although the aforementioned methods mitigate the impact of dynamic targets to some extent, issues such as low detection efficiency and substantial computational resource requirements persist.

2.2 Extraction and Tracking of Feature Points

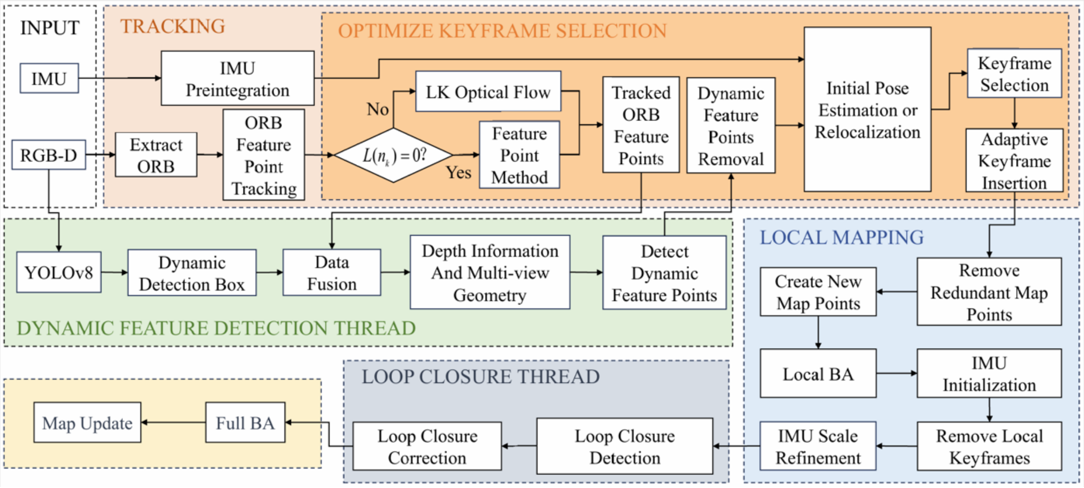
Feature point tracking in SLAM algorithms is crucial for subsequent localization and mapping. Common feature point extraction and tracking algorithms include SIFT, SURF, and ORB [10]. Fu et al. [11] proposed using corners extracted by the FAST algorithm as feature points, then describing these feature points using the SIFT algorithm. Hsieh et al. [12] used image frames captured via SURF matching to compute the distance from environmental feature points to the origin of the camera coordinate system, thereby obtaining the relative position of the vehicle. The SIFT and SURF algorithms are relatively mature but suffer from poor real-time performance. The ORB-SLAM series represents a classic implementation of the ORB algorithm, with ORB-SLAM3 capable of extracting feature points from key regions such as corners and edges. This algorithm operates stably and exhibits minimal sensitivity to changes in lighting conditions [13]. However, calculating the FAST corners and BRIEF descriptor for each frame individually is time-consuming, reducing the overall efficiency of the algorithm. Therefore, after initially extracting ORB feature points, this algorithm employs optical flow for inter-frame tracking of these points, thereby avoiding the computation of descriptors for every frame.

3. System Description

3.1 System Framework

The YDK-SLAM system proposed by this algorithm comprises four threads: the tracking thread, the dynamic feature detection thread, the local mapping thread, and the loop closure thread. These four threads operate in parallel logically. The system architecture is shown in Figure 1. The tracking thread performs ORB feature extraction, tracking, and matching on data captured by the RGB-D camera to obtain ORB feature points. Meanwhile, the YOLOv8 network within the dynamic feature detection thread performs preliminary detection of moving objects within the RGB-D signal. This data is then fused with the ORB feature points to distinguish between dynamic and static points. Subsequently, depth information and multi-view geometry methods are employed to further detect and eliminate dynamic feature points, retaining only static points. These static points are combined with pre-integrated IMU signals to achieve pose estimation and tracking. To prevent tracking failures caused by loss of scene information during sudden camera movements while maintaining computational efficiency, we have optimized the selection of key frames. When the camera moves smoothly, the optical flow method is used to track feature points, eliminating the time required to compute matching descriptors and effectively improving tracking speed. When camera motion becomes more intense, the feature point method is employed for matching and tracking to prevent tracking loss and enhance the accuracy of the tracking thread. Adaptively insert keyframes generated by the tracking thread into the local mapping thread, prune redundant map points, create new map points, and perform local bundle adjustment optimization on the map. To eliminate systematic errors in the IMU and provide stable initial states, the pre-integrated IMU parameters are initialized using maximum a posteriori estimation. After removing local keyframes and performing scale fine-tuning, the data is fed into the loop closure thread for loop closure detection and loop closure correction, addressing the issue of pose estimation drifting over time. Finally, global BA is employed to optimize all map points and keyframes, enabling map updates.

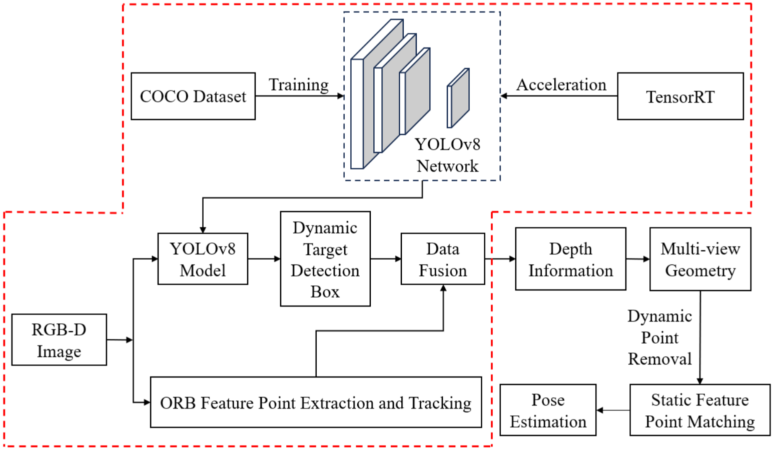
Figure 1: System architecture diagram



3.2 Preliminary Detection of Dynamic Feature Points

The dynamic object detection and removal module is shown in Figure 2. First pre-trains the YOLOv8 model using the MS COCO dataset. Objects are categorized into a priori dynamic objects and a priori static objects based on their type. The former refers to objects with high dynamic probability and unstable poses, such as people, cats, and dogs in indoor environments. The latter encompasses all other objects not classified as a priori dynamic objects, including chairs, cups, books, keyboards, and so on. The TensorRT module is employed to accelerate the training process.

Figure 2: Dynamic object detection and removal module



The RGB-D image is then fed into the trained YOLOv8 network to detect all objects in the current frame, yielding the set of object detection bounding boxes X^k. The set of ORB feature points extracted by the tracking thread is denoted as Y^k.

$$X^k = \{x_1^k, x_2^k, x_3^k, ..., x_m^k\},\ Y^k = \{y_1^k, y_2^k, y_3^k, ..., y_n^k\} \tag{1}$$

Where x_m^k denotes the m-th detection box in the k-th frame image, and y_n^k denotes the n-th feature point in the k-th frame image. If the object within detection box x_m^k is a human or animal, and y_n^k is located within detection box x_m^k, then y_n^k is classified as a prior dynamic point, otherwise, it is classified as a static point. Based on the above comprehensive judgment, the dynamic feature point set $P_{dynamic} = \{p_{d1}, p_{d2}, p_{d3}, ...\}$ and the static feature point set $P_{static} = \{p_{s1}, p_{s2}, p_{s3} .....\}$ are obtained.

3.3 Dynamic Feature Point Elimination Based on Depth Information

Object bounding boxes detected by YOLOv8 often overlap with background elements, potentially misclassifying static background features as dynamic ones and causing tracking failures. Therefore, we utilize depth information from RGB-D cameras to segment dynamic targets from the background via depth thresholding. In the background, a prior dynamic object typically exhibits a noticeable depth difference compared to a prior static object. Assuming d represents depth, if a pixel's depth is available, then $d > 0$. Pixels for which no valid depth is acquired are marked as $d = 0$. For each detected moving object, such as a person, extract the four corner points of its bounding box. The maximum background depth ${}^{q}d_{max}$ of the q-th dynamic object in the object detection box set $X^k = \{x_1^k, x_2^k, x_3^k, ..., x_m^k\}$ is defined as:

$$ {}^{q}d_{max} = \max\left({}^{q}d_{tl}, {}^{q}d_{tr}, {}^{q}d_{bl}, {}^{q}d_{br}\right) \tag{2}$$

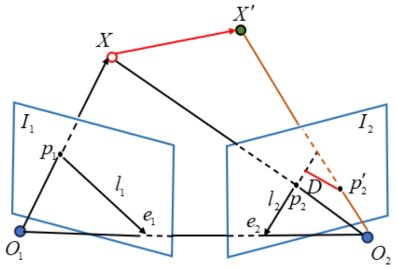
${}^{q}d_{tl}, {}^{q}d_{tr}, {}^{q}d_{bl}, {}^{q}d_{br}$ respectively denote the depth of the top-left, top-right, bottom-left, and bottom-right corner points of the object bounding box. Suppose the center depth value of the bounding box where the q-th dynamic point is located is ${}^{q}d_{c}$, then the depth threshold is determined by the maximum depth of the corner points of the object detection box and the depth value at the center of the box, and is defined as:

$$ {}^{q}\overline{d} = \begin{cases} \frac{1}{2}\left({}^{q}d_{\max} + {}^{q}d_{c}\right), & \text{if } {}^{q}d_{\max} - {}^{q}d_{c} > \varepsilon, {}^{q}d_{c} > 0 \\ {}^{q}d_{c} + \varepsilon, & \text{if } {}^{q}d_{\max} - {}^{q}d_{c} < \varepsilon, {}^{q}d_{c} > 0 \\ {}^{q}d_{\max}, & \text{if } {}^{q}d_{\max} > 0, {}^{q}d_{c} = 0 \\ +\infty, & \text{otherwise} \end{cases} \tag{3}$$

$\varepsilon > 0$ is a predefined distance based on the most common dynamic objects in the environment. The first scenario occurs when the depth difference between a moving object and its background is significant. In this case, the threshold is set to the midpoint between the background and object depths, and a semantic mask is applied between them to separate the moving target from the background. The second scenario occurs when the dynamic target is closely integrated with the background depth or positioned behind the background. In this case, the depth threshold is set at a distance ε from the dynamic target, where a semantic mask is generated to prevent the background from being misclassified as a dynamic point. The third scenario is when only the maximum background depth exists, in which case the depth threshold is set to the maximum background depth value. If none of the first three scenarios apply, infinite depth is used as the threshold. Compare each feature depth d in the RGB-D image with the corresponding depth threshold $\overline{d}$ on the semantic mask. If $d < \overline{d}$, it is classified as a dynamic feature point, otherwise, it is considered a static feature point. When depth information becomes invalid, it leads to increased depth errors, which in turn affects the calculation of depth thresholds. Therefore, it is necessary to further filter and eliminate dynamic points.

3.4 Movement Consistency Check

Figure 3: Multi-view geometric constraint relationships



To avoid the issue of depth information becoming invalid, this algorithm combines multi-view geometry algorithms with the use of depth information alone to further remove dynamic points. Multi-view geometric constraints are shown in Figure 3. O_1 and O_2 are the optical centers of the camera at two different moments when it moves. The intersection points e_1 and e_2 of image planes I_1 and I_2 with baseline $O_1\ O_2$ are called epipoles. The intersection lines of the epipolar plane XO_1O_2 with the image planes are called epipolar lines, denoted by l_1 and l_2.

If the spatial point X is a static point, then its image point p_1 on I_1 and image point p_2 on I_2 must lie in the epipolar plane, and p_2 must be constrained to the epipolar line l_2. If X is a dynamic point, it does not satisfy this constraint. The specific calculation method is as follows:

The coordinates of the corresponding pixel feature matching points in the known image frame are $p_1 = [u_1, v_1]$ and $p_2 = [u_2, v_2]$. The homogeneous coordinates of a pixel can be expressed as $P_1 = [u_1, v_1, 1]$ and $P_2 = [u_2, v_2, 1]$, the fundamental matrix F represents the geometric relationship of camera motion. If the spatial point X is a static point, it satisfies the epipolar constraint condition.

$$P_2^T F P_1 = 0 \tag{4}$$

Epipolar line l_2 is:

$$l_2 = FP_1 = [A, B, C]^T \tag{5}$$

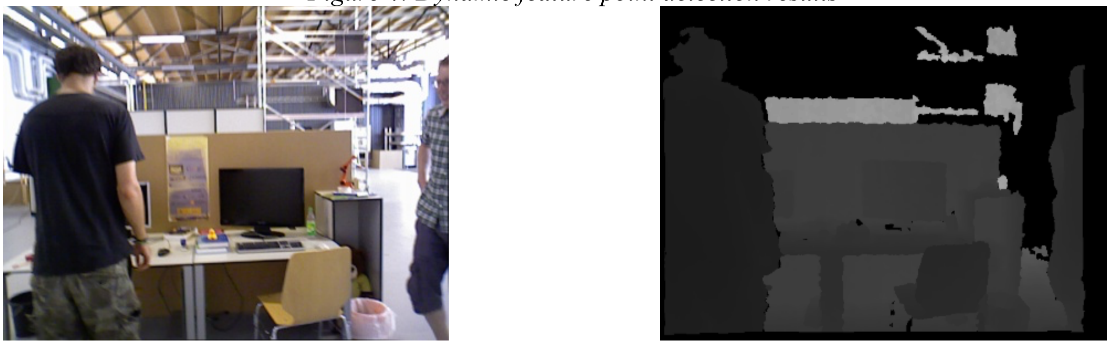
where $[A, B, C]^T$ represents the direction vector of epipolar line l_2.

If X is a dynamic point and moves to point X', its pixel position changes to p_2' at this time, which deviates from the epipolar line l_2 and does not satisfy the constraint condition. Therefore, the distance from p_2' to the epipolar line l_2 can be used to determine dynamic points. The coordinates of the pixel feature point p_2' are defined as $p_2' = [u_2', v_2']$, and after homogeneous transformation, they become $P_2' = [u_2', v_2', 1]$. The distance from p_2' to the epipolar line l_2 is:

$$D = \frac{|P_2' F P_1^T|}{\|l_2\|} = \frac{|P_2' F P_1^T|}{\sqrt{A^2 + B^2}} \tag{6}$$

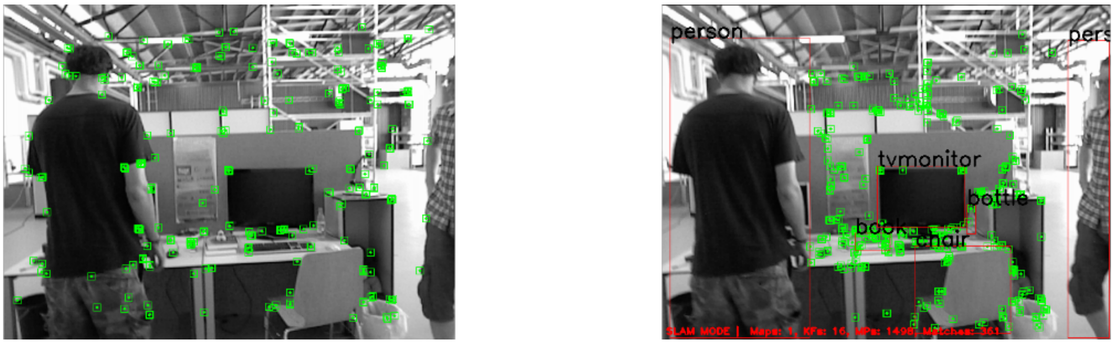
Finally, a threshold τ is set. If the distance D from a feature point in the object detection box to the corresponding epipolar line is greater than τ, the feature point is filtered out, while static feature points are retained, static feature points are then used for map initialization and tracking. The results of dynamic feature point detection are shown in Figure 4. The green dots represent detected feature points, while the red boxes denote dynamic object detection boxes. Figure 4(c) is the detection result of ORB-SLAM3. In the area of the dynamic object (person), there are still unremoved dynamic feature points. While Figure 4(d) is the detection result of the algorithm in this paper, which accurately identifies and effectively removes the dynamic feature points, and retains the static feature points.

Figure 4: Dynamic feature point detection results



(a) Original Image

(b) Deep Image



(c) ORB-SLAM3 results

(d) YDK-SLAM results

3.5 Optimized Keyframe Extraction

The ORB-SLAM3 algorithm commonly employs a feature-based visual odometer for tracking feature points, calculating FAST corners and BRIEF descriptors frame-by-frame for extracted points. This extensive redundant computation process results in relatively low overall algorithmic efficiency. Therefore, this algorithm proposes a more flexible tracking strategy based on this foundation. Firstly, the ORB feature extraction is performed on the input image, and the feature points are directly tracked using the optical flow method. Then, the key frame selection strategy is determined based on the evaluation function $L(n_{\mathrm{k}})$. The evaluation function is as shown in Formula 7. Define the ratio of the number of feature points tracked by the current optical flow method to the number tracked in the previous frame as $p\left(n_k/n_{k-1}\right)$, where σ is an empirical threshold. When $L(n_{\mathrm{k}})=1$, continue to use the optical flow tracking algorithm. When $L(n_{\mathrm{k}})=0$, it indicates that the environment changes relatively drastically, so exit the optical flow-based visual odometry and switch to the feature point-based visual odometry to select keyframes.

$$L(n_{\mathrm{k}})=\begin{cases}1, & p\left(n_k/n_{k-1}\right)<\sigma\\ 0, & p\left(n_k/n_{k-1}\right)>\sigma\end{cases} \tag{7}$$

For tracked feature points, YDK-SLAM triggers keyframe insertion when any of the following conditions are met while inserting new keyframes into the existing ORB-SLAM3 framework:

(1) When the number of frames since the last keyframe insertion reaches the maximum threshold, it indicates that no keyframes have been inserted for an extended period. At this point, the current image frame shares minimal common visual regions with the previous keyframe, potentially leading to tracking loss. Therefore, a new keyframe is inserted.

(2) When a local thread is idle and at least the minimum frame interval has elapsed since the last keyframe insertion, a new keyframe is inserted. This effectively prevents excessive image overlap and ensures idle local mapping threads have sufficient capacity to process new keyframes.

(3) In the local mapping module, to reduce computational load and minimize information redundancy, the number of key frames in the key frame queue shall not exceed three.

4. Experimental Results and Analysis

4.1 Evaluation Criteria

The experimental platform for this paper is a Linux system running Ubuntu 20.04, with the following hardware configuration: Intel® Core™ i7-14650HX processor, 24GB of memory, and an NVIDIA GeForce RTX 4060 graphics card. The TUM dataset provided by the Technical University of Munich (TUM) is utilized as experimental data. This dataset contains various sensor inputs, such as RGB cameras, depth cameras, and IMUs, for evaluating different types of algorithms. This study employed six dynamic datasets for testing: fr3/sitting_xyz, fr3/sitting_halfsphere, fr3/walking_static, fr3/walking_xyz, fr3/walking_rpy, and fr3/walking_halfsphere. Among these, fr3/sitting_xyz and fr3/sitting_halfsphere represent low-dynamic scenes where subjects are seated or moving slowly, with relatively gradual camera movements and environmental changes. In contrast, the fr3/walking dataset features high-dynamic scenes where subjects are continuously walking or engaged in various activities, resulting in more complex camera motions.

The TUM dataset provides real-time camera position and pose information, enabling the estimation of camera trajectories by running different algorithms in the same environment. The EVO evaluation tool is employed to analyze the obtained experimental data, using absolute trajectory error (ATE) and relative pose error (RPE) as evaluation metrics for comparative experiments. In this algorithm, three metrics—the median error, mean error, and root mean square error (RMSE) of the ATE—serve as standards for evaluating accuracy. Among these, RMSE is a commonly used statistical measure to quantify the discrepancy between the camera's estimated trajectory and the actual trajectory. A smaller RMSE indicates a smaller deviation between the estimated and actual values, signifying higher trajectory estimation accuracy. Conversely, a larger RMSE indicates lower trajectory estimation accuracy.

4.2 Comparison Experiment on Localization Accuracy with ORB-SLAM3

The YDK-SLAM algorithm presented in this paper is an improvement upon ORB-SLAM3, hence a performance comparison between the two algorithms is conducted. During experimentation, each of the six selected datasets was run at least three times, with results averaged. Here, we define the algorithm improvement rate as ρ, calculated using the formula:

$$\rho = \frac{B - A}{B} \times 100\% \tag{8}$$

Here, B represents the results of the ORB-SLAM3 algorithm, while A denotes the optimized results from this algorithm. Table 1 below compares the absolute trajectory error (ATE) before and after optimization, while Table 2 compares the relative pose error (RPE) before and after optimization.

Table 1: Absolute trajectory error results for orb-slam3 algorithm and proposed algorithm

Sequence	ORB-SLAM3/m			YDK-SLAM/m			ρ/%		
	Median	Mean	RMSE	Median	Mean	RMSE	Median	Mean	RMSE
sitting_xyz	0.0079	0.0104	0.0119	0.0088	0.0098	0.0104	-11.39	5.76	12.60
sitting_halfsphere	0.0356	0.0358	0.0396	0.0177	0.0224	0.0250	50.28	37.43	36.86
walking_static	0.0634	0.0763	0.0926	0.0055	0.0060	0.0067	91.32	92.13	92.76
walking_xyz	0.2200	0.2399	0.2788	0.0098	0.0112	0.0131	95.54	95.33	95.30
walking_rpy	0.3742	0.4038	0.4560	0.0232	0.0269	0.0328	93.80	93.34	92.80
walking_halfsphere	0.2820	0.3062	0.3376	0.0142	0.0168	0.0200	94.96	94.51	94.07

Table 2: Relative pose error results for orb-slam3 algorithm and proposed algorithm

Sequence	ORB-SLAM3/m			YDK-SLAM/m			ρ/%		
	Median	Mean	RMSE	Median	Mean	RMSE	Median	Mean	RMSE
sitting_xyz	0.0063	0.0075	0.0092	0.0069	0.0071	0.0084	-9.52	5.34	8.69
sitting_halfsphere	0.0095	0.0104	0.0118	0.0095	0.0124	0.0135	0.00	-19.2	-14.04
walking_static	0.0040	0.0049	0.0062	0.0015	0.0024	0.0038	47.5	36.73	29.03
walking_xyz	0.0076	0.0093	0.0113	0.0071	0.0079	0.0093	6.57	15.05	19.41
walking_rpy	0.0156	0.0203	0.0233	0.0119	0.0157	0.0179	23.71	22.67	23.17
walking_halfsphere	0.0121	0.0172	0.0203	0.0093	0.0109	0.0137	23.14	31.97	32.51

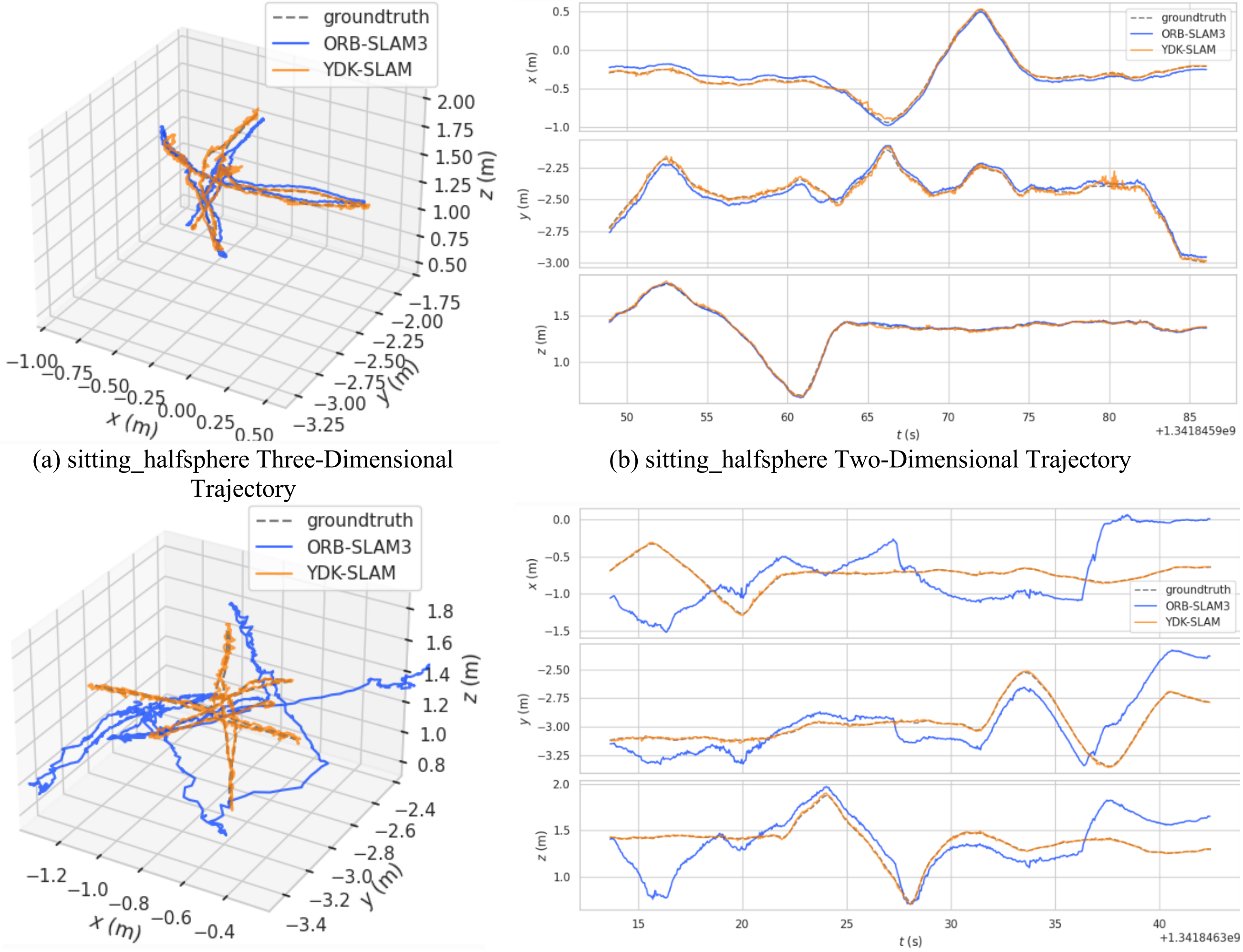
As shown in Tables 1 and 2, when operating in low-dynamic environments, the ORB-SLAM3 algorithm demonstrates superior performance with smaller positioning accuracy errors compared to camera-estimated trajectories. However, in high-dynamic environments, it exhibits significant tracking loss, leading to increased positioning errors. The YDK-SLAM algorithm maintains positioning accuracy comparable to ORB-SLAM3 in low-dynamic scenes, with some improvements observed. In high-dynamic environments, the YDK-SLAM algorithm achieves an average reduction of 93.73% in absolute trajectory root mean square error compared to ORB-SLAM3. Overall, all accuracy metrics across the six dynamic datasets improved and outperformed ORB-SLAM3, with the RMSE of ATE decreasing by an average of 70.73%. This indicates that the YDK-SLAM algorithm achieves enhanced positioning accuracy after removing interference from dynamic objects, with more pronounced improvements in highly dynamic scenes.

Figure 5 shows the comparison results between the estimated trajectories and true trajectories of the ORB-SLAM3 and YDK-SLAM algorithms across three datasets: fr3/sitting_halfsphere, fr3/walking_xyz, and fr3/walking_halfsphere. The gray dashed line represents the camera's true trajectory, the blue solid line indicates the estimated trajectory from ORB-SLAM3, and the orange solid line denotes the estimated trajectory from the proposed algorithm. Figure 5(a) and Figure 5(b) show the 3D trajectory map and 2D trajectory map in low-dynamic scenes, respectively. Comparison reveals that both the ORB-SLAM3 algorithm and the proposed algorithm exhibit high estimation accuracy. However, in the high-dynamic scenes depicted in Figure 5(c), Figure 5(d), Figure 5(e), and Figure 5(f), the YDK-SLAM algorithm's estimated camera trajectory more closely approximates the true trajectory, demonstrating superior performance. In contrast, the ORB-SLAM3 algorithm's estimated trajectory exhibits significant deviation and low overlap with the true trajectory.

4.3 Comparison of Positioning Accuracy with Other Dynamic Slam Algorithms

Comparative experiments were conducted between the YDK-SLAM algorithm and other dynamic visual SLAM algorithms across four high-dynamic datasets. The experimental results are presented in Table 3. As shown in the table, when compared to current representative dynamic visual SLAM algorithms such as DS-SLAM and DynaSLAM, the YDK-SLAM algorithm exhibits the smallest camera trajectory error on high-dynamic datasets. By evaluating the root mean square error (RMSE) of absolute trajectory error, the YDK-SLAM algorithm consistently outperforms both DS-SLAM and DynaSLAM algorithms.

Figure 5: Trajectory estimation diagram for ORB-SLAM3 and the algorithm proposed in this paper

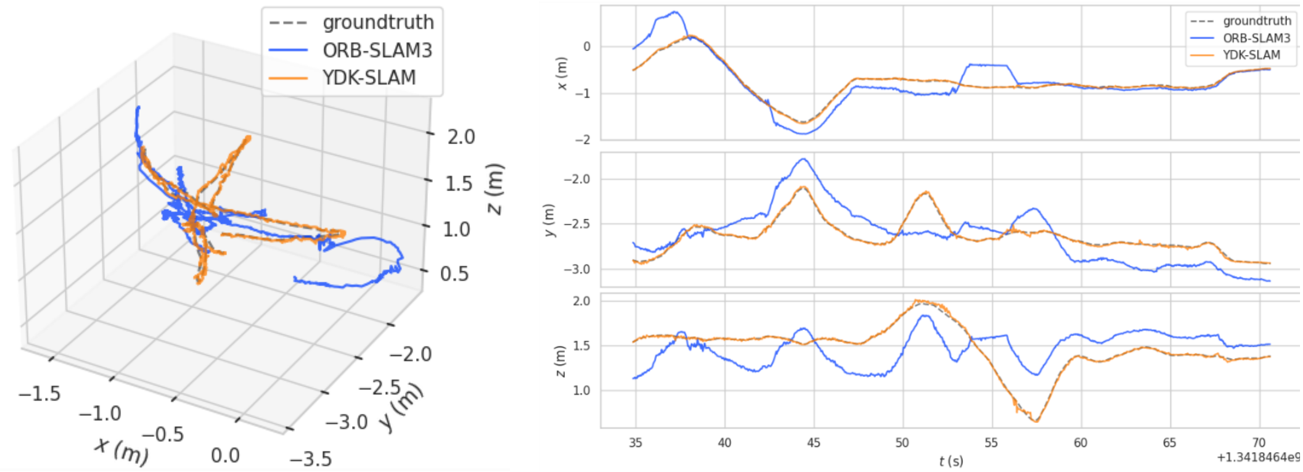


(a) sitting_halfsphere Three-Dimensional Trajectory

(b) sitting_halfsphere Two-Dimensional Trajectory

(c) walking_xyz Three-Dimensional Trajectory

(d) walking_xyz Two-Dimensional Trajectory



(e) walking_halfsphere Three-Dimensional Trajectory

(f) walking_halfsphere Two-Dimensional Trajectory

Table 3: Comparison of RMSE for absolute trajectories between the proposed algorithm and other dynamic slam algorithms

Sequence	DS-SLAM/m	DynaSLAM/m	YDK-SLAM/m
walking_static	0.0081	0.0068	0.0067
walking_xyz	0.0247	0.0170	0.0131
walking_rpy	0.4143	0.0354	0.0328
walking_halfsphere	0.0300	0.0269	0.0200

4.4 Comparison of Real-time Performance with Other Dynamic Slam Algorithms

To validate the real-time performance of the proposed algorithm, comparative experiments were conducted against ORB-SLAM3 and DynaSLAM using three high-dynamic datasets. The experimental results are shown in Table 4, the YDK-SLAM algorithm achieves higher real-time performance than DynaSLAM. However, due to the added object detection process, it requires more time than ORB-SLAM3, yet its overall processing time remains relatively short.

Table 4: Comparison of computational time for different algorithms on high-dynamic datasets

Sequence	ORB-SLAM3/s	DynaSLAM/s	YDK-SLAM/s
walking_xyz	0.0252	1.022	0.1002
walking_rpy	0.0249	1.051	0.0933
walking_halfsphere	0.0255	1.107	0.1071

5. Conclusion

To address issues such as trajectory tracking failures and reduced positioning accuracy in dynamic environments, this paper proposes the YDK-SLAM algorithm. Building upon the original ORB-SLAM3 algorithm, it not only incorporates a dynamic feature detection thread but also employs depth thresholding and multi-view geometry methods to further eliminate dynamic features. First, the YOLOv8 object detection network is employed to identify moving objects. Depth information is utilized to segment backgrounds and moving targets through depth thresholding. When depth information extraction fails, geometric motion consistency is leveraged to further eliminate potential false dynamic features. Next, we propose optimizing keyframe selection to prevent tracking loss and enhance system performance. The proposed algorithm was tested using dynamic sequences from the TUM dataset. Experimental results demonstrate that compared to the ORB-SLAM3 algorithm, the YDK-SLAM algorithm achieves an average accuracy improvement of 70.73% across six datasets. Under high-dynamic conditions, the RMSE of the proposed algorithm decreases by over 90% in all aspects, exhibiting excellent performance. This algorithm offers insights for visual SLAM in removing dynamic objects, but its performance under extreme conditions such as low-light environments and texture-deficient surfaces remains untested. Additionally, the algorithm incurs a certain computational cost. Further research could focus on refining feature point selection and matching, as well as developing lightweight object detection networks, to enhance matching accuracy and reduce computational demands.

References

[1] Placed J A, et al. A survey on active simultaneous localization and mapping: State of the art and new frontiers[J]. IEEE Transactions on Robotics, 2023, 39(3): 1686-1705.

[2] Campos C, Elvira R, Rodríguez J J G, Montiel J M M, Tardós J D. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM[J]. IEEE Transactions on Robotics, 2021, 37(6): 1874-1890.

[3] Yu C, et al. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments[C]. Madrid, Spain: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018: 1168-1174.

[4] Bescos B, Fácil J M, Civera J, Neira J. DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes[J]. IEEE Robotics and Automation Letters, 2018, 3(4): 4076-4083.

[5] Pan Z, Hou J, Yu L. Optimization RGB-D 3-D Reconstruction Algorithm Based on Dynamic SLAM[J]. IEEE Transactions on Instrumentation and Measurement, 2023, 72: 1-13.

[6] Cheng S, Sun C, Zhang S, Zhang D. SG-SLAM: A Real-Time RGB-D Visual SLAM Toward Dynamic Scenes With Semantic and Geometric Information[J]. IEEE Transactions on Instrumentation and Measurement, 2023, 72: 1-12.

[7] Su P, Luo S, Huang X. Real-Time Dynamic SLAM Algorithm Based on Deep Learning[J]. IEEE Access, 2022, 10: 87754-87766.

[8] Chang Z, Wu H, Li C. YOLOv4-tiny-based robust RGB-D SLAM approach with point and surface feature fusion in complex indoor environments[J]. Journal of Field Robotics, 2022, 40(3): 521-534.

[9] Wen S, Tao S, Liu X, Babiarz A, Yu F R. CD-SLAM: A Real-Time Stereo Visual–Inertial SLAM for Complex Dynamic Environments With Semantic and Geometric Information[J]. IEEE Transactions on Instrumentation and Measurement, 2024, 73: 1-8.

[10] Fu J, Ma P, Jia Z. An Improved Monocular ORB-SLAM Using Scaled-Invariant Features[C]. Wulumuqi, China: IEEE International Conference on Advanced Information, Mechanical Engineering, Robotics and Automation (AIMERA), 2024: 125-130.

[11] Fu X, Zhang K, Shen C, Zhu J, Zang L. Recognition and Location Based on Fusion of FAST Algorithm and SIFT Algorithm[C]. Tianjin, China: IEEE 21st International Conference on Communication Technology (ICCT), 2021: 422-426.

[12] Hsieh J-W, Chen L-C, Chen D-Y. Symmetrical SURF and Its Applications to Vehicle Detection and Vehicle Make and Model Recognition[J]. IEEE Transactions on Intelligent Transportation Systems, 2014, 15(1): 6-20.

[13] Zu L, Wei C, Sun Q, Zhang M, Sheng N, Ge S S. Adaptive Keyframe Selection Strategy of Visual SLAM in Complex Poses[J]. IEEE Sensors Journal, 2025, 25(1): 1756-1767.

Funding

This paper was supported by Henan Provincial Engineering Technology Research Center for Optoelectronic Detection and Sensor Integration (Jiaozuo, Henan, China).

Conflicts of Interest

The authors declare no conflict of interest.

Acknowledgment

This paper is an output of the science project.

Copyrights